REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 18-04-2012	**2. REPORT TYPE** Final Technical	**3. DATES COVERED** *(From - To)* 08-15-2010 to 02-14-2012
4. TITLE AND SUBTITLE (U) CONTINUED FUNDING FOR PRIME DEVELOPMENT		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER FA9550-10-1-0450
		5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Michael Frenklach		**5d. PROJECT NUMBER** 2308
		5e. TASK NUMBER BX
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of California at Berkeley, Department of Mechanical Engineering, Berkeley, CA 94720-1740		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research, 875 North Randolph Street, Suite 325, Room 3112, Arlington, VA 22203-1768		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-OSR-VA-TR-2012-1016

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The initiative named PrIMe (for Process Informatics Model) is designed to keep track of models, model parameters, and experimental data in a global, integrated framework for the field of Combustion. It is aimed at curation of community data with the objective of collaborative development of reaction mechanisms of scientific explorations and predictive models for practical systems. The present project was a continuation of a prior AFOSR grant (FA9550-08-1-0003, Program Manager: Dr..Julian Tishkoff), which enabled, among other things, initial development of one of the principal PrIMe components, PrIMe Workflow Application. The additional funding under the present AFOSR Grant allowed us to bring this development to its stable operational version, Workflow 2.0. Also, with this additional support we added new scientific tools to the Workflow. In this Report outline our past-year accomplishments and describe the details of the PrIMe Workflow 2.0 release.

15. SUBJECT TERMS

Modeling, combustion, global systems, web-based application, collaborative science

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Chiping Li
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified		66	**19b. TELEPHONE NUMBER** *(include area code)* (703) 696-8574

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Continued Funding for PrIMe Development

Michael Frenklach

Final Technical Report to AFOSR
April 18, 2012

Department of Mechanical Engineering
University of California
Berkeley, CA 94720

Table of contents

1 Introduction

The objective of this project was to sustain and extend the development of the PrIMe cyber-infrastructure (CI) for the practical use by the Combustion community. PrIMe (Process Informatics Model) is a new approach for developing predictive models of chemical reaction systems that is based on the scientific collaboratory paradigm. The primary goals of PrIMe are collecting and storing data, validating the data and quantifying uncertainties, and assembling the data into predictive models with quantified uncertainties to meet specific user requirements. The principal elements of PrIMe include: a data Warehouse which is a repository of data provided by the community, a data Library which archives community-evaluated data, and computer-based tools to process data and to assemble data into predictive models.

Optimizing combustion efficiency and understanding the mechanisms that prevent full energy utilization of fuels relies on detailed knowledge of the underlying physics and chemistry. These systems are generally complex enough that models have been used to explore the effect of different feed and reactor conditions and have been successful in optimizing fuel mixtures and combustor performance. However, the models are extremely complex and often controversial. The data, which parameterize the models and are compared to model predictions, are themselves complex and often open to interpretation. Further, they are developed by multiple labs using different technologies. To keep track of models, parameters, and data in an integrated framework has proven a necessity in the field of Combustion. The PrIMe initiative is designed to fill this need. In its scientific content, PrIMe is a *system approach* aimed at establishing the infrastructure, both scientific and CI, in support of developing *predictive* models of combustion.

The initial phase and development of PrIMe CI has focused on underlying chemical reaction models. There are several important reasons for this strategy. First, modeling of a combustion process begins with a reaction model, which determines the concentrations of chemical specifies and the heat flux, and hence it is only natural to start the new development from this founding stage. It has been our experience[1] that most disagreements between models and experiments and most controversies begin with and trace to the selected reaction model. Another factor for starting with reaction models is the fact that chemical kinetics has accumulated much needed data and the missing data can be evaluated using quantum and reaction-rate theories. And finally, the scientific underpinning of the process, also illustrating the feasibility of the approach, has been piloted by the GRI-Mech project.

The present project was a continuation of a prior AFOSR grant (FA9550-08-1-0003, Program Manager: Dr. Julian Tishkoff), which enabled, among other things, initial development of one of the principal PrIMe components, PrIMe Workflow Application. The additional funding under the present AFOSR Grant allowed us to bring this development to its stable operational version, Workflow 2.0. Also, with this additional support we added new scientific tools to the Workflow.

In this Report, we first outline our past-year accomplishments, and then describe the details of the PrIMe Workflow 2.0 release.

2 Past Year Accomplishments and Current Status of PrIMe

The PrIMe infrastructure has the following principal elements, a Data Warehouse, Tools, and Workflow, as well as a community Portal. During the past year we made progress in all these areas.

2.1 PrIMe Portal

The PrIMe Portal is based on the Drupal open-source software. During the past year we upgraded it to version 6.

There are currently over 350 registered users and over 20 Work groups.

During the past year we developed many new video tutorials on operation of the PrIMe CI components.

2.2 PrIMe Data Warehouse

The PrIMe Data Warehouse has been populated with:

- over 100,000 records of data on chemical species, species thermodynamics, chemical reactions, reaction rate coefficients, reaction models
- over 400 records of experimental data related to combustion, collected in shock tubes, flow reactions, and laminar premixed flames: ignition delays, species profiles, flame speeds, soot
- during the past year, in collaboration with Markus Kraft of Cambridge University, UK, and JoAnn Lighty of the University of Utah, a large collection of data on soot formation

2.3 PrIMe Workflow Application

The PrIMe Workflow is a centerpiece of the PrIMe cyber-infrastructure. It links data and apps and enables the users to conduct their research activities in a “menu-driven”, web-based operation. Building it was made possible by employing a professional programmer, which, in turn, was made possible by the MACCCR funding received from Dr. Julian Tishkoff in 2008 and a continuation of it with the present Grant.

During the past year, we developed and implemented Version 2.0 of the PWA. It is built on a cloud-based model and currently offers:

- a much more stable operation
- faster response time
- support for different types of applications; those written in C# and Matlab run on client machines and those implemented in any other way can run on a remote server linked via built for this purpose PrIMe web services
- novel scientific applications, those of uncertainty quantification
- redesigned menu-driven component submission interface

2.4 Building Predictive Combustion Models through PrIMe

The ultimate goal and purpose of PrIMe is to support development of truly predictive combustion models. During the past year, we added two new scientific tools: DataCollaboration, an on-line application for systematic uncertainty quantification, and most recently, a sensitivity-analysis tool, currently limited to shock-tube ignition. We completed the first global system for on-demand model building: the user can now build, on the fly, a reaction model for hydrogen combustion that meets his/her specific conditions/requirements, or an experimenter can test whether his/her new results (on H_2/O_2) are consistent with and complement the existing data, and how much improvement is gained in predictiveness of the hydrogen-combustion model overall. We published a manuscript describing this: "Process Informatics Tools for Predictive Modeling: Hydrogen Combustion", X. You, A. Packard, M. Frenklach, Int. J. Chem. Kinet. 44:101-116, 2012.

2.5 PrIMe Instrumental Model

During the past year, we continued our joint project with Professor Phillip Westmoreland's NCSU group on the PrIMe Instrumental Data Model, aiming at a systematic approach to dealing with the "raw" experimental data. The user is asked to submit not only his/her data (raw or processed, experimental or theoretical) but also "describe" the exact procedure used to "process" these data. The PrIMe data-management software will capture and archive this information in a computer-readable form. The initial development was done using a "simpler" case, using the shock-tube ignition data, and the code is now in place. During the past year, we extended this work to a more complex case, analysis of data collected in a fuel-lean $C_2H_2/O_2/Ar$ premixed laminar flat flame, mapped with VUV-photoionization molecular-beam mass spectrometry at the Advanced Light Source of Lawrence Berkeley National Laboratory. The experimental signals were modeled with a premixed laminar flat-flame code augmented with an Instrumental Model, designed to link raw signals to derived properties. The consistency of the model and raw experimental data are quantified, and features of the mole fraction profiles for weak-signal observations of O, OH, C_2H_3 and unknown background H_2O are predicted. The approach to model-versus-data assessment demonstrated in this study promises to advance the science and practical utility of modeling, establishing validity rigorously while identifying and ranking the impacts of specific model and data uncertainties. A manuscript summarizing these results ("Integrated Analysis of Acetylene-Flame Data and Model Uncertainties Using an Instrumental Model Approach", D. R. Yeates, W. Li, P. R. Westmoreland, T. Russi, A. Packard, and M. Frenklach) is in preparation for publication.

3 PrIMe Workflow Architecture

The PrIMe Workflow Application (PWA) is a web-based application that unifies the components of PrIMe into a single interface. The purpose of this document is to describe the PrIMe Workflow Application architecture and its internal structure. The functionality of PWA components is depicted in the form of Use Case diagrams. Class diagrams, consistency diagrams, data-base scheme, and components diagrams are used to demonstrate system design and component interaction.

The document describes the following aspects of the PrIMe Workflow Application architecture:

1. The common system structure and the purpose of its modules.
2. PrIMe portal description and functionality.
3. Component Uploader general architecture description and functionality.
4. PrIMe Workflow Application general architecture description and functionality.

4 The common system structure

The common structure of PrIMe is shown in Figure 1. It consists of the following components:

1. **PrIMe Portal** is responsible for system user management. It implements user authorization and authentication services, assigns user roles, and manages user permissions. Additionally, it enables users to collaborate. In the PrIMe portal one can find information such as the latest changes, documentation, and operating instructions. The Development Portal provides an interface for two additional systems—Scientific Component Uploader and PrIMe Workflow Application.

2. **Scientific Component Uploader (SCU)** is used to develop and deploy new scientific components. It allows the scientific component developer to upload a new scientific component, assign resources to components, and edit properties and configuration information of his/her previously developed components. All changes made by developers are stored as separate revisions, allowing a developer to open and edit any existing revisions. Only a user with administrator privileges can create a new revision and deploy it to the PrIMe Workflow Application (PWA).

3. **PrIMe Workflow Application (PWA)** is the "environment" where a user creates and executes scientific workflow projects. The scientific workflow project is built using preconfigured scientific components that are linked together in a network. The user can set input and output information for each scientific component and, if applicable, the user

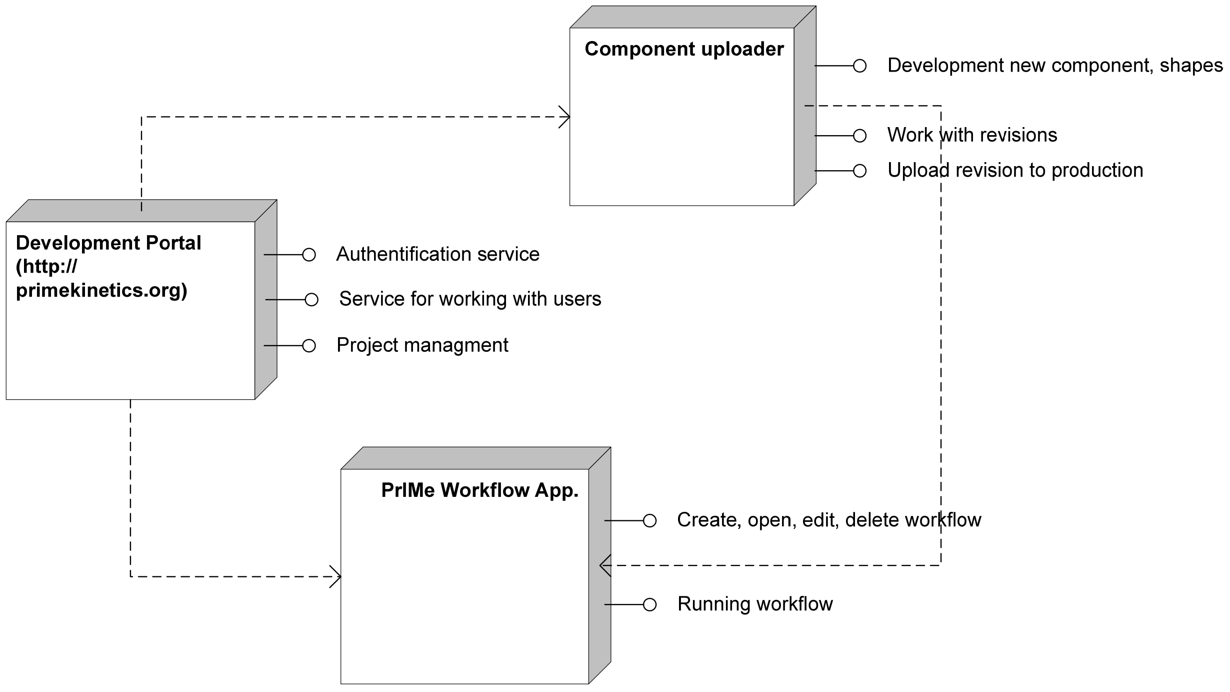

Component uploader
Development new component, shapes
Work with revisions
Upload revision to production
Development Portal (http:// primekinetics.org)
Authentification service
Service for working with users
Project managment
PrIMe Workflow App.
Create, open, edit, delete workflow
Running workflow

Figure 1. Components of the PrIMe structure

can set configurable properties of a scientific component. In the PrIMe Workflow Application a user can create new scientific workflow projects, open existing projects, and execute valid workflow projects.

5 PrIMe Portal

Purpose

The purpose of the PrIMe Portal is to administer user management functions and stores workflow project information. The main system Use Case is represented in Figure 2.

System functions:

1. *User management functionality*. The PrIMe Portal implements user authentication and authorization. Additionally, it assigns roles to each user and grants permissions to view/edit the user's previously existing workflow projects and scientific components.

2. *Content management functionality*. The PrIMe Portal manages all application documentation, which includes the scientific components manuals, system structure changes, and information concerning development of new scientific components.

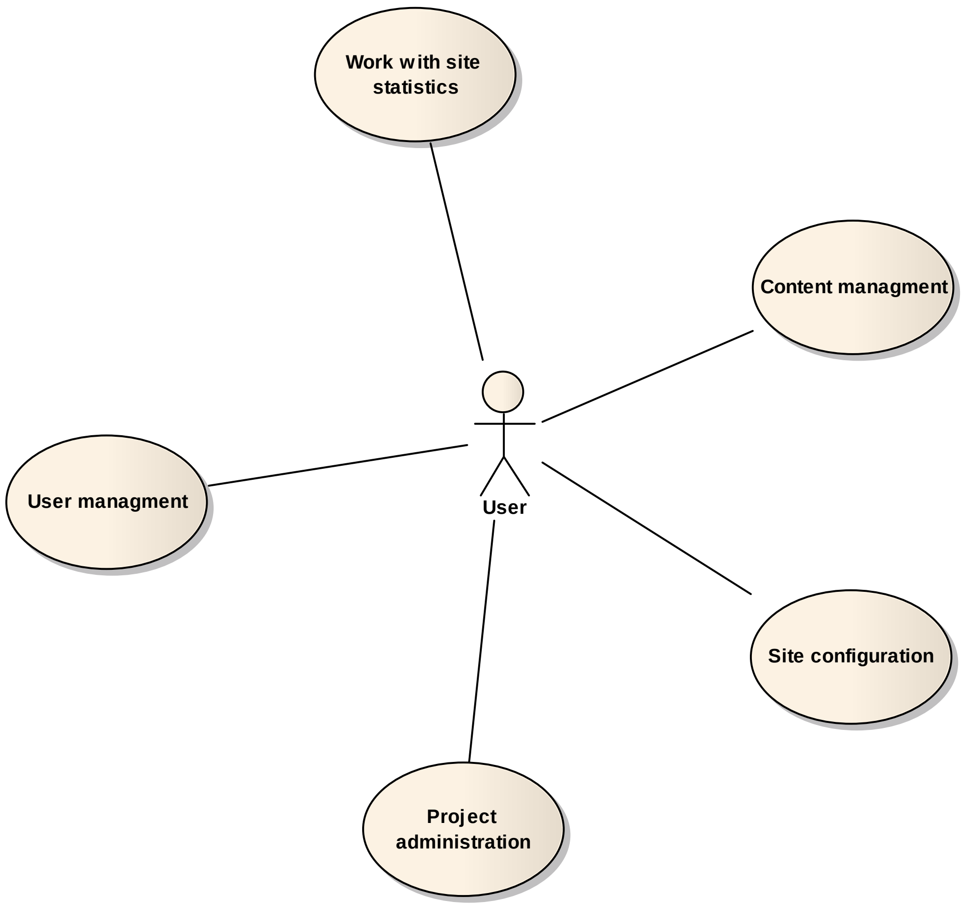


Figure 2. PrIMe portal Use Case diagram

3. *Authentication/authorization management for PWA and SCU*. The PrIMe Portal authenticates, authorizes, and assigns user roles allowing a user to access PWA and SCU. This management is done automatically as the user navigates to the PWA and SCU.

6 System architecture of the SCU and PWA

Architecture presents on the figure 2. It consists of the next things:

1. SCU and PWA were implemented as ClickOnce application.
2. Backend server includes business logic implemented in the core, data-access layer and provides WCF-services for communication with client. It hosts inside asp.net web application.
3. WCF services. Provide all necessary functionality for working with workflows, components and so on.
4. Back-end code. Includes data-access layer for interaction with database, business logic, AA logic.
5. Also, there is clickOnce application which includes UI for PWA and SCU and will deploy on the client machine. When client opens page in his browser, clickOnce application deploys on the client machine and runs, using this application user can manage components via SCU or work with workflows.

There are next benefits why clickOnce technology was selected

1. User doesn't need to install application, just go to the web-site and launch it.
2. Upgrade is very easy and performs every time when user runs application, if there is new version, of course.

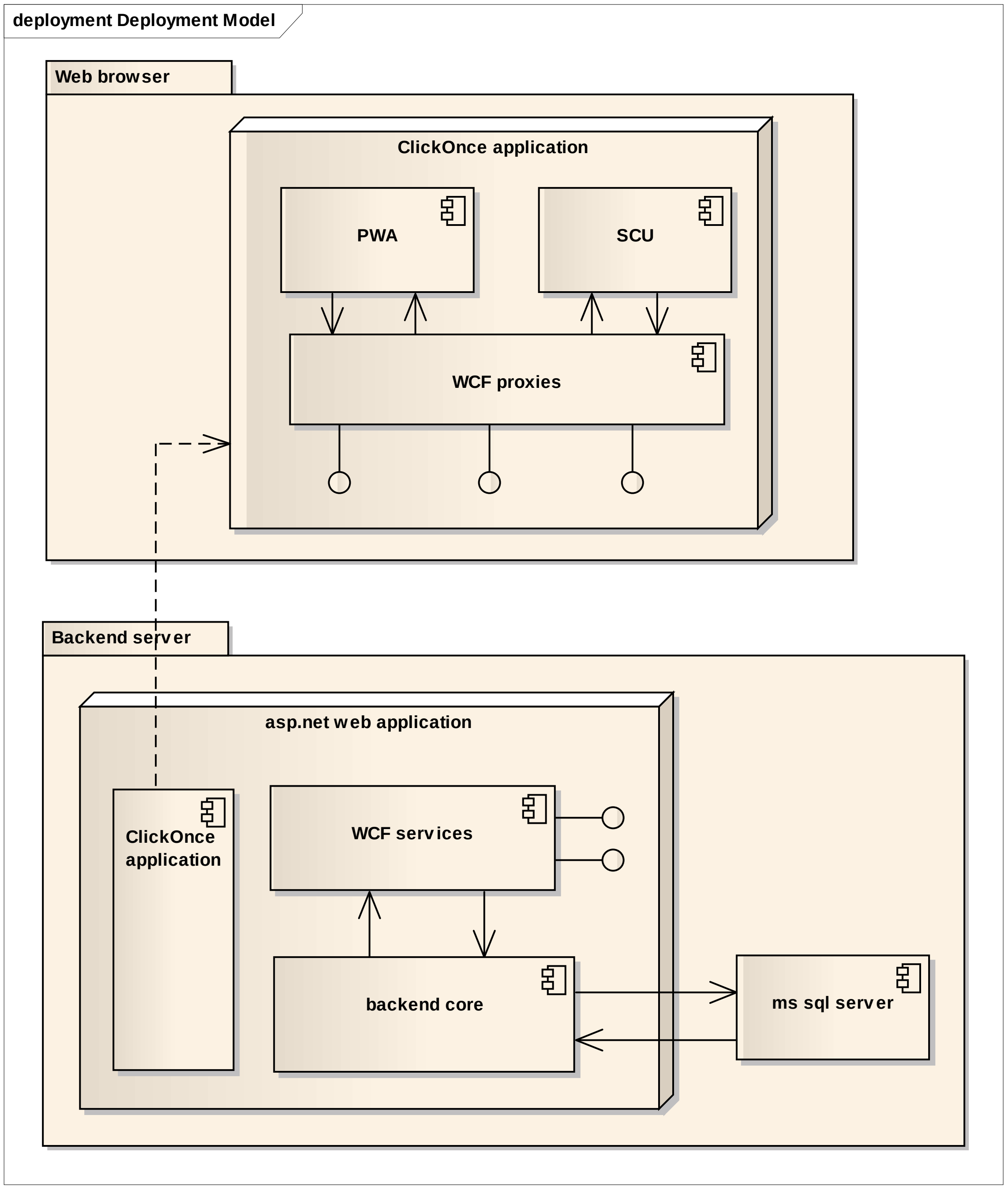


Figure 2. System architecture of the PWA and SCU.

7 Scientific Component Uploader (SCU)

The main purpose of the Scientific Component Uploader is to facilitate the development and deployment of scientific components. It enables development of scientific components by managing existing component revisions, component properties, component resources, and configuration information and allows creation of new scientific components. The Scientific

Component Uploader also allows a developer to test the component before it is deployed to the PWA.

7.1 Use Case systems

The SCU manages scientific components, resources, and component configuration information. The main functions of the SCU are listed below, and the SCU Use Cases are represented in Figure 4.

Main functions:

1. *Manage component resources*. The user can upload new component images, edit images, and remove existing images associated with a scientific component from the server.
2. *Manage scientific components*. The SCU enables the user to add, remove, and edit scientific components.
3. *Configure scientific components*. The user can configure scientific components by adding, editing, and removing component inputs, outputs, and other properties.
4. *Store scientific component location*. If the scientific component is a remote type, the SCU points to the server from which the remote scientific component is executed.
5. *Manage component revisions*. The SCU captures and saves all changes made by the user when editing components, resources, and configuration information as revisions. The MS SQL server stores all revision information.
6. *Scientific component testing*. The SCU allows the scientific component developer to test his/her component and confirm that it will work appropriately with PWA.
7. *Scientific component deployment*. The SCU allows an administrator to deploy any revision of a scientific component to the PWA.

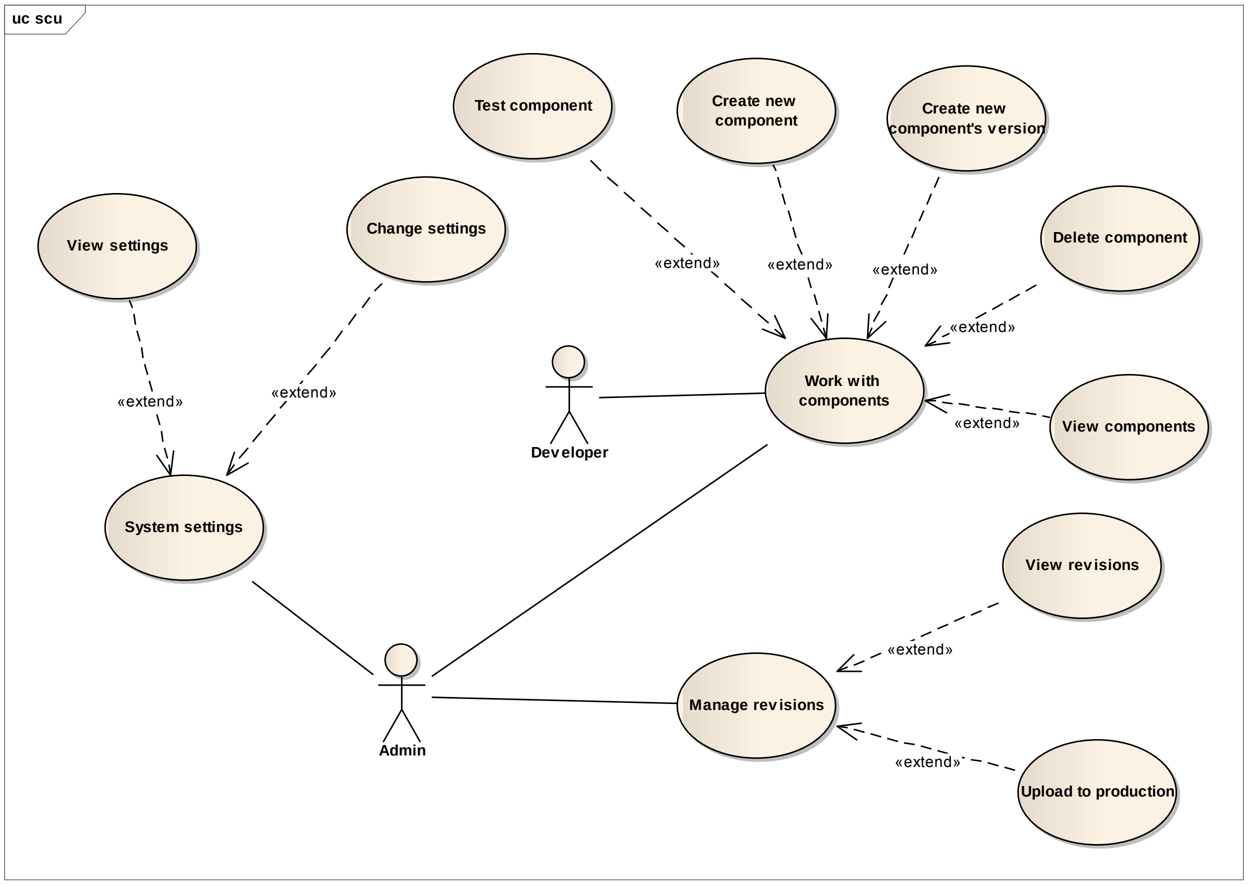


Figure 4. Use Case of the SCU

7.2 SCU clickonce application structure

In the clickOnce application the actions related to resources, components, configuration information, and scientific component deployment into production.

7.2.1 Main SCU modules

The library structure is represented in Figure 5.

The library consists of the following modules:

1. *Images*—The module to manage resources. It provides operations to add new resources or delete resources from the server.
2. *Components*—The module to manage scientific components. It provides operations to add components, edit components, and delete components.
3. *Libraries*—The module for working with libraries. It provides functions to add, edit, or delete libraries.

4. *Revisions*—The module to manage system revisions. It provides the creation of new revisions, and deletion and deployment of selected revisions to the PWA production server.
5. *Settings – service for managing system preferences(matlab preferences, hdf viewer and so on)*

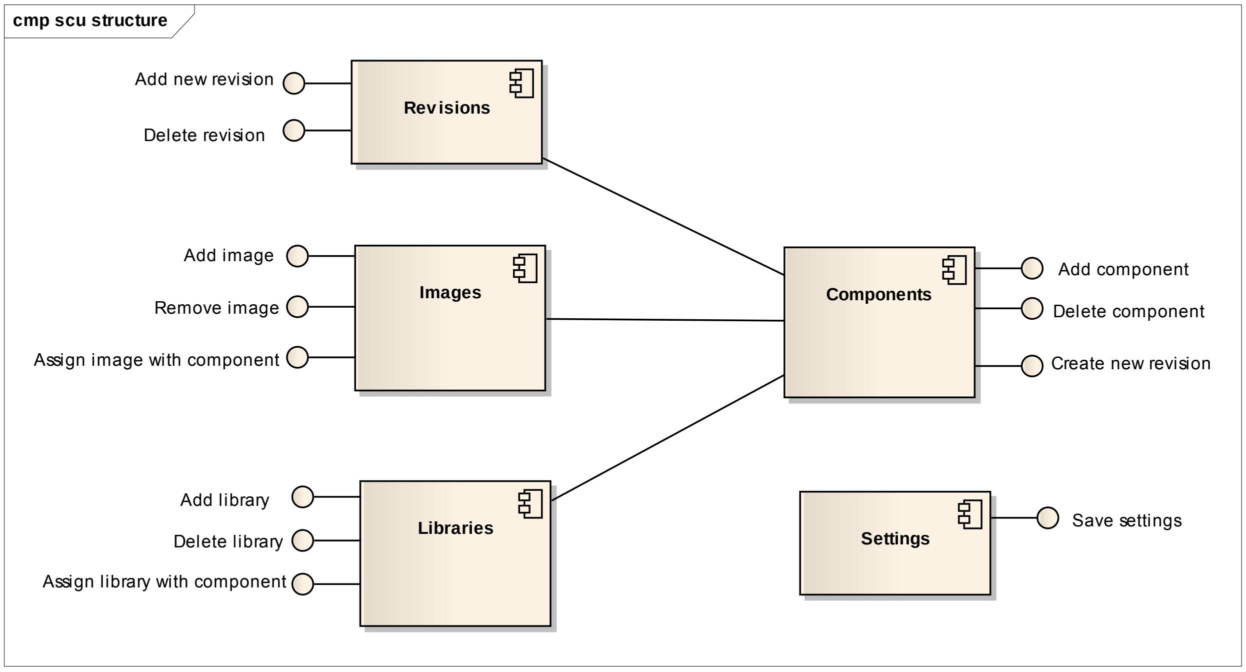


Figure 5. SCU structure

7.2.2 SCU clickOnce classes diagram

The main SCU clickOnce classes diagram is shown in Figure 6.

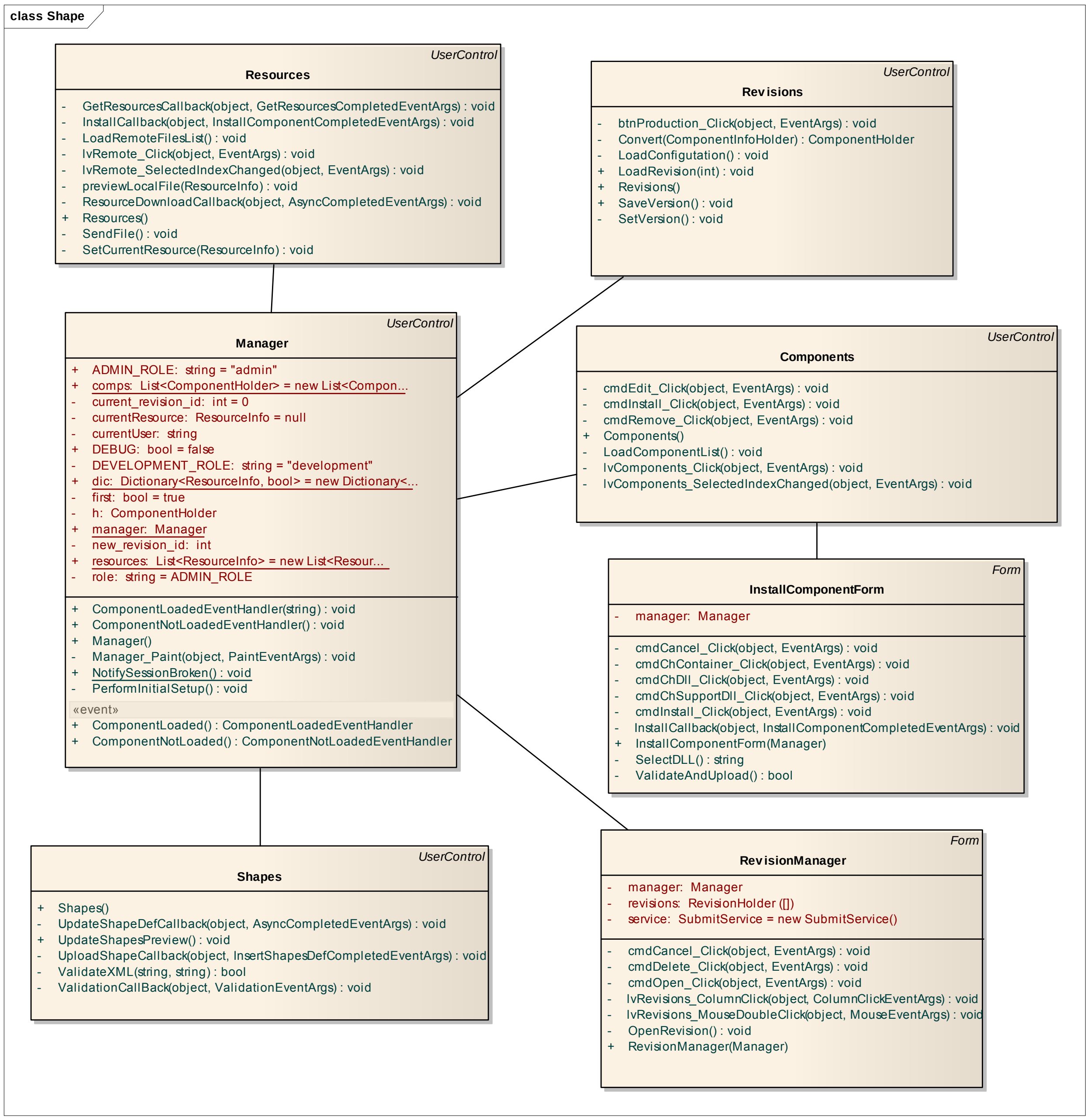

class Shape
UserControl
Resources
- GetResourcesCallback(object, GetResourcesCompletedEventArgs) : void
- InstallCallback(object, InstallComponentCompletedEventArgs) : void
- LoadRemoteFilesList() : void
- lvRemote_Click(object, EventArgs) : void
- lvRemote_SelectedIndexChanged(object, EventArgs) : void
- previewLocalFile(ResourceInfo) : void
- ResourceDownloadCallback(object, AsyncCompletedEventArgs) : void
+ Resources()
- SendFile() : void
- SetCurrentResource(ResourceInfo) : void
UserControl
Revisions
- btnProduction_Click(object, EventArgs) : void
- Convert(ComponentInfoHolder) : ComponentHolder
- LoadConfigutation() : void
+ LoadRevision(int) : void
+ Revisions()
+ SaveVersion() : void
- SetVersion() : void
UserControl
Manager
+ ADMIN_ROLE: string = "admin"
+ comps: List<ComponentHolder> = new List<Compon...
- current_revision_id: int = 0
- currentResource: ResourceInfo = null
- currentUser: string
+ DEBUG: bool = false
- DEVELOPMENT_ROLE: string = "development"
+ dic: Dictionary<ResourceInfo, bool> = new Dictionary<...
- first: bool = true
- h: ComponentHolder
+ manager: Manager
- new_revision_id: int
+ resources: List<ResourceInfo> = new List<Resour...
- role: string = ADMIN_ROLE
+ ComponentLoadedEventHandler(string) : void
+ ComponentNotLoadedEventHandler() : void
+ Manager()
- Manager_Paint(object, PaintEventArgs) : void
+ NotifySessionBroken() : void
- PerformInitialSetup() : void
«event»
+ ComponentLoaded() : ComponentLoadedEventHandler
+ ComponentNotLoaded() : ComponentNotLoadedEventHandler
UserControl
Components
- cmdEdit_Click(object, EventArgs) : void
- cmdInstall_Click(object, EventArgs) : void
- cmdRemove_Click(object, EventArgs) : void
+ Components()
- LoadComponentList() : void
- lvComponents_Click(object, EventArgs) : void
- lvComponents_SelectedIndexChanged(object, EventArgs) : void
Form
InstallComponentForm
- manager: Manager
- cmdCancel_Click(object, EventArgs) : void
- cmdChContainer_Click(object, EventArgs) : void
- cmdChDll_Click(object, EventArgs) : void
- cmdChSupportDll_Click(object, EventArgs) : void
- cmdInstall_Click(object, EventArgs) : void
- InstallCallback(object, InstallComponentCompletedEventArgs) : void
+ InstallComponentForm(Manager)
- SelectDLL() : string
- ValidateAndUpload() : bool
UserControl
Shapes
+ Shapes()
- UpdateShapeDefCallback(object, AsyncCompletedEventArgs) : void
+ UpdateShapesPreview() : void
- UploadShapeCallback(object, InsertShapesDefCompletedEventArgs) : void
- ValidateXML(string, string) : bool
- ValidationCallBack(object, ValidationEventArgs) : void
Form
RevisionManager
- manager: Manager
- revisions: RevisionHolder ([])
- service: SubmitService = new SubmitService()
- cmdCancel_Click(object, EventArgs) : void
- cmdDelete_Click(object, EventArgs) : void
- cmdOpen_Click(object, EventArgs) : void
- lvRevisions_ColumnClick(object, ColumnClickEventArgs) : void
- lvRevisions_MouseDoubleClick(object, MouseEventArgs) : void
- OpenRevision() : void
+ RevisionManager(Manager)

Figure 6. SCU classes

A summary of the classes available in SCU is given below.

Manager—This class implements main GUI library

Method	Assignment
Manager_Paint	This method activates at window repainting.
PerformInitial	Activates once at library download. In this method the revisions for the current user are downloaded, and the library download process is displayed.
NotifySessionBroken	Informs the user about the errors which occurred during the process.
componentLoaded	Downloads component.

Resources—In this class the work with resources is implemented. This class directly processes user's activities.

Method	Assignment
LoadRemoteFileList	Downloads the resource list from the server
SendFile	Downloads the file to the server
GetResourceCallBack	Responds after the resources have completed downloading to the server

InstallComponentsForm—This class implements the GUI for working with components

Method	Assignment
ValidateandUpload	Validates the input data and saves the components
cmdInstall_Click	Processes the request to save the component
cmdChDll_Click	Processes the request to select the component from the library
InstallComponentForm	Adds a new component

Components—The work with components is implemented in this class

Method	**Assignment**
LoadComponentList	Downloads a list of the components
cmdInstall_Click	Opens the form to input component information
LvComponents_Click	Processes the selection of components from the list. Outputs the detailed information about the component

Revisions—The work with the revisions is implemented in this class

Method	**Assignment**
btnProduction_Click	Processes the requests to download revisions to PWA
LoadRevision	Downloads the specified revision
saveVersion	Saves revisions

Shapes—The work with shapes is implemented in this class

Method	**Assignment**
ValidateXml	Checks xml-description of shapes
UpdateShapesPreview	Refreshes shapes after updating
UploadShapesCallBack	Stores shapes

RevisionManager—The GUI which works with revisions

Method	**Assignment**
cmbDelete_Click	Deletes revisions
cmdOpen_Click	Opens a specified revision
lvRevision_Click	Selects the revisions

7.3 Backend structure

In the library user authorization and authentication management is implemented. Additionally the core implements server side functionality of the SCU by storing scientific components, configuration information, and resources in the database. Figure 7 shows a schematic of the backend structure.

7.3.1 Main part of SCU backend

The library consists of the following modules:

1. *Authentication module*. Its main functions are to receive the user credentials, to get the users roles, and to start a new session for the user.
2. *Revision module*. The main functions of the revision module are to get a revision by id, delete a revision, identify the current revision, and add a new revision.
3. *Configuration Information (Shapes) Service*. The library provides such functions as gets shapes on revision number, paste the new shape, and get shape by id.
4. *Component module*. The main functions of the component module are to get all of the scientific components, to get the scientific components by revision, to add a new scientific component, and to update the scientific component.

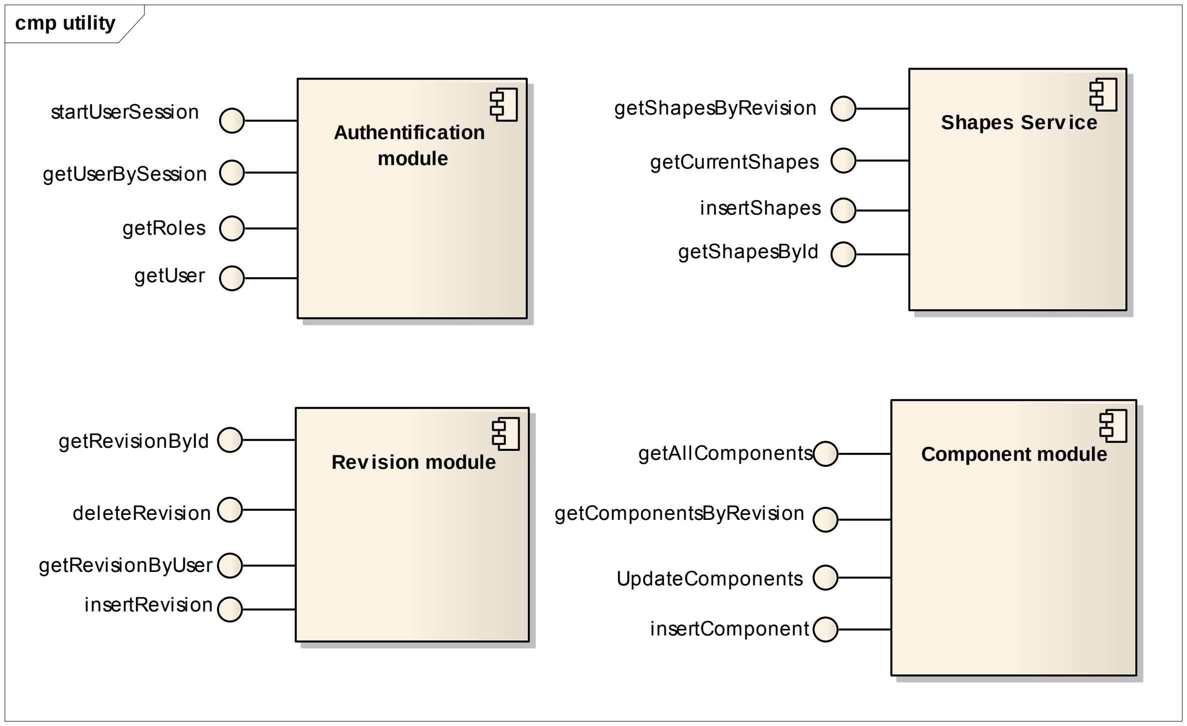


Figure 7. Utility.dll structure

7.3.2 Back end classes diagram

The structure of classes diagram is shown on Figure 8. The library consists of the following main classes.

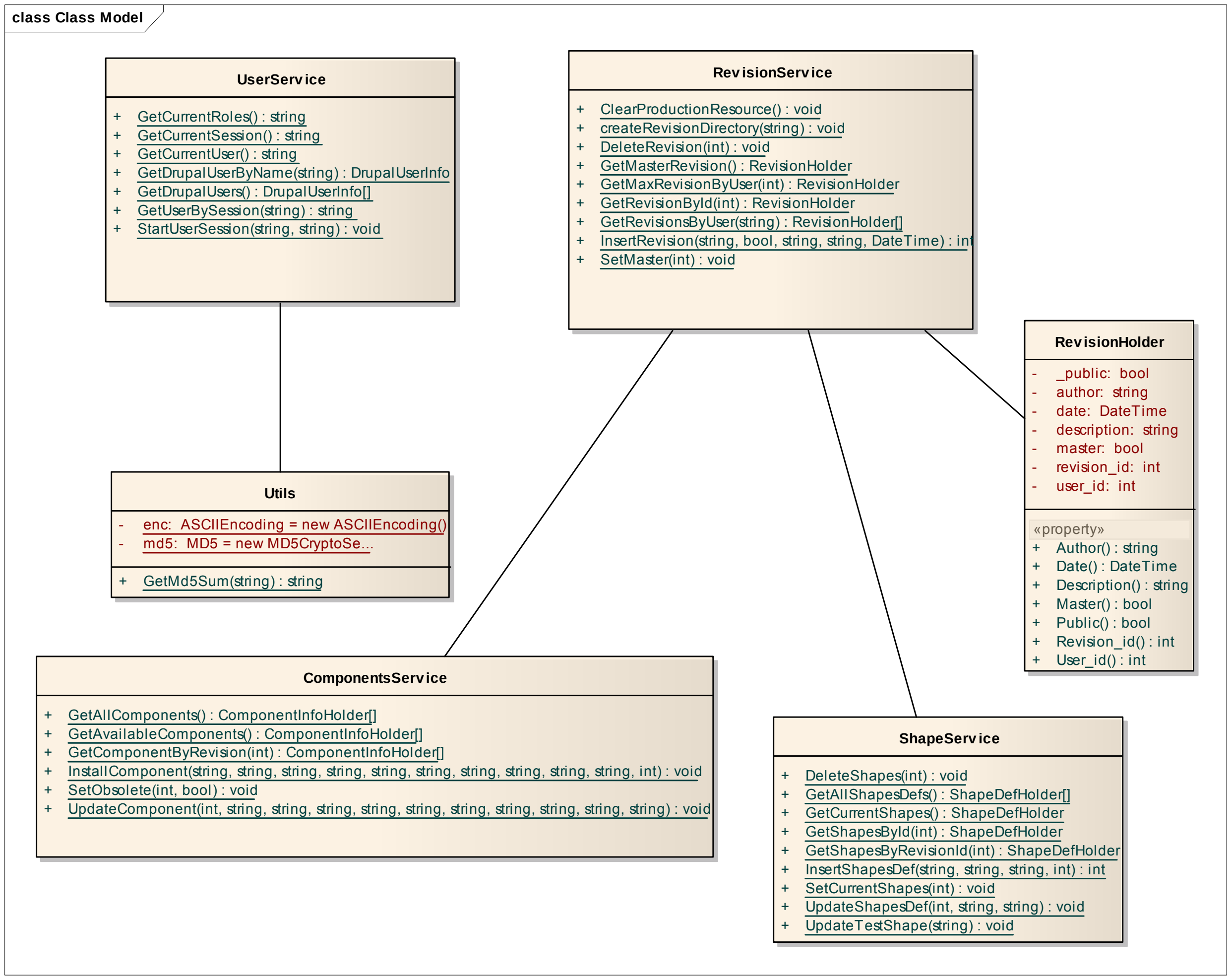

class Class Model
UserService
+ GetCurrentRoles() : string
+ GetCurrentSession() : string
+ GetCurrentUser() : string
+ GetDrupalUserByName(string) : DrupalUserInfo
+ GetDrupalUsers() : DrupalUserInfo[]
+ GetUserBySession(string) : string
+ StartUserSession(string, string) : void
RevisionService
+ ClearProductionResource() : void
+ createRevisionDirectory(string) : void
+ DeleteRevision(int) : void
+ GetMasterRevision() : RevisionHolder
+ GetMaxRevisionByUser(int) : RevisionHolder
+ GetRevisionById(int) : RevisionHolder
+ GetRevisionsByUser(string) : RevisionHolder[]
+ InsertRevision(string, bool, string, string, DateTime) : int
+ SetMaster(int) : void
RevisionHolder
- _public: bool
- author: string
- date: DateTime
- description: string
- master: bool
- revision_id: int
- user_id: int
«property»
+ Author() : string
+ Date() : DateTime
+ Description() : string
+ Master() : bool
+ Public() : bool
+ Revision_id() : int
+ User_id() : int
Utils
- enc: ASCIIEncoding = new ASCIIEncoding()
- md5: MD5 = new MD5CryptoSe...
+ GetMd5Sum(string) : string
ComponentsService
+ GetAllComponents() : ComponentInfoHolder[]
+ GetAvailableComponents() : ComponentInfoHolder[]
+ GetComponentByRevision(int) : ComponentInfoHolder[]
+ InstallComponent(string, string, string, string, string, string, string, string, string, string, int) : void
+ SetObsolete(int, bool) : void
+ UpdateComponent(int, string, string, string, string, string, string, string, string, string, string) : void
ShapeService
+ DeleteShapes(int) : void
+ GetAllShapesDefs() : ShapeDefHolder[]
+ GetCurrentShapes() : ShapeDefHolder
+ GetShapesById(int) : ShapeDefHolder
+ GetShapesByRevisionId(int) : ShapeDefHolder
+ InsertShapesDef(string, string, string, int) : int
+ SetCurrentShapes(int) : void
+ UpdateShapesDef(int, string, string) : void
+ UpdateTestShape(string) : void

Figure 8. SCU backend classes

UserService— The service for work with users is implemented in this class

Method	**Assignment**
GetCurrentRoles	Returns the current user roles
GetCurrentSession	Returns the session of the current user
GetDrupalUsers	Returns all the users who are registered in the system
GetUserBySession	Returns the users on session
StartUserSession	Starts the new session for the specified user

RevisionService—The service for work with revisions

Method	Assignment
CreateRevisionDirectory	Creates a new catalog, where all the revisions and the library components will be stored
DeleteRevision	Deletes of the specified revision
GetMaxRevisioByUser	Returns the last revision that the user edited
InsertRevision	Inserts a new revision
GetRevisionById	Returns a revision by specified id

ComponentService—The service for work with components

Method	Assignment
GetAllComponents	Returns all the components
getComponentsByRevision	Returns the components by revision
UpdateComponents	Updates the information about the specified component
InstallComponent	Adds a new component

ShapeService—The service for working with shapes

Method	Assignment
DeleteShapes	Deletes a specified shape
GetAllShapesDef	Returns all shapes
InsertShapesDef	Adds new shape descriptions
UpdateShapesDef	Updates a specified shape
GetShapesByRevision	Returns the shapes description for a specified revision
GetShapesById	Returns the shape description by specified id

7.4 Database structure

The database structure is shown in Figure 9.

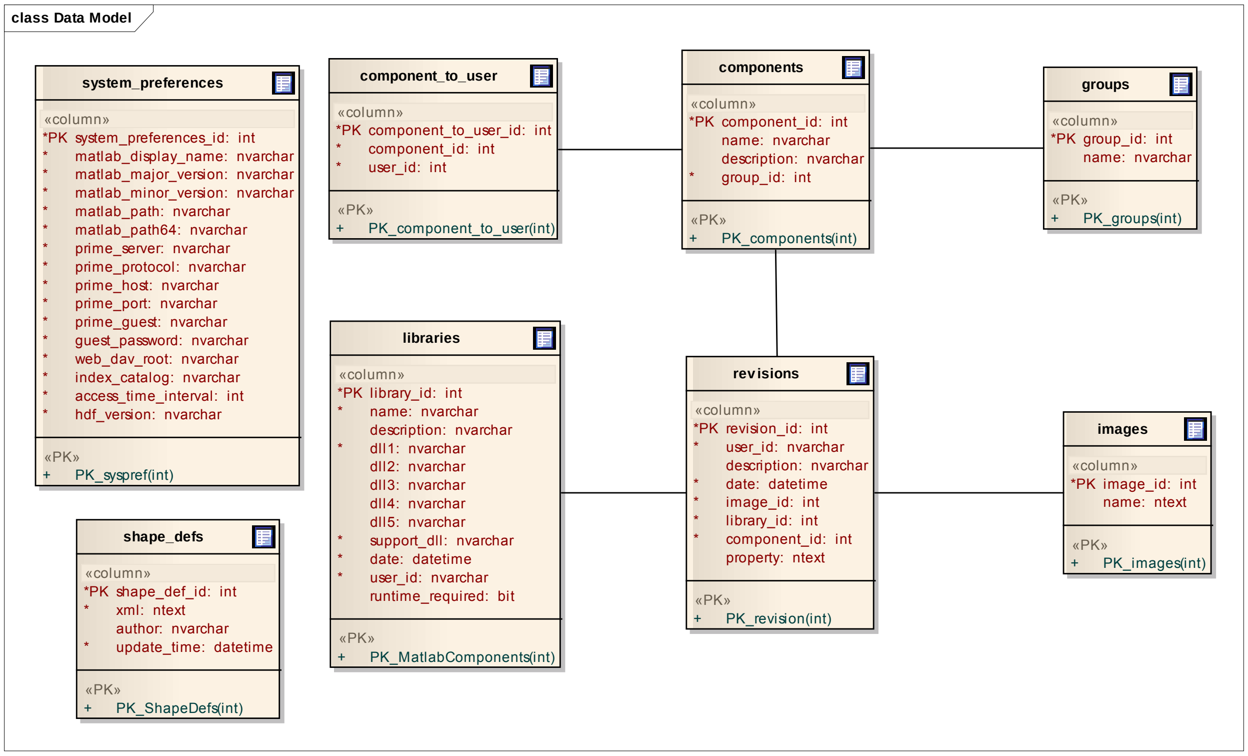


Figure 9. Database structure.

Revisions—Stores the revisions created by the users

Field	**Description**
revision_id	Revision identifier
user_id	User identifier, who stored this revision
description	Revision description
date	Creation date
image_id	Reference on the image
library_id	Reference on the library
component_id	Reference on the component

ShapeDefs—Stores the shape descriptions

Field	**Description**
ShapeDefId	Shape description identifier
Xml	Xml-description of the shape
Description	Shape description
Author	The author who created the shape
updateTime	The update time
Revision_id	The revision identifier, to which the shape refers

Libraries—Stores libraries information

Field	**Description**
library_id	Unique id
Name	Library name
Description	Component description
Dll1, Dll2, Dll3, Dll4, Dll5	Libraries names
SupportDll	Library name that provides the interface of the connection with the PWA
Date	Creation date
User_id	User who last updated library
Runtime_required	Indicates if component requires matlab runtime

Images—Stores images

Field	**Description**

Image_id	Unique id
name	Image name

Groups—Stores groups

Field	Description
group_id	Unique id
name	Group name

Components—Stores components

Field	Description
component_id	Unique id
name	Component name
description	Component description
Group_id	Reference on group

Component_to_user—Stores permissions on the components

Field	Description
component_to_user_id	Unique id
Component_id	Reference on component
User_id	Reference on user

System_preferences—Stores system settings

Field	Description
System_preference_id	Unique id
Matlab_display_name	Name of current matlab version in registry
Matlab_major_version	Current matlab major version
Matlab_minor_version	Current matlab minor version
Matlab_path	Path to current x86 matlab runtime
Matlab_path64	Path to current x64 matlab runtime
Prime_server	Warehouse server
Prime_protocol	Protocol for communication with warehouse server
Prime_host	IP or DNS warehouse server
Prime_port	port
Prime_guest	Login to warehouse server

Guest_password	Password to warehouse server
Web_dav_root	Root path on the web_dav server
Access_time_interval	Allowed interval for connection

7.5 Web services

Web services provide a convenient method of communication between client and server via HTTP protocol. A short description of main methods is presented below.

Method	**Description**
GetAllComponents	Returns all the components from the server
GetShapesById	Returns shapes by id
GetResources	Returns resources by revision
GetLastRevisionToUser	Returns the last revision for the user
InstallComponent	Adds new components
InsertShapesDef	Adds new shapes
GetShapesByRevision	Returns shapes by resource identifier
GetComponentByRevision	Returns components by revision identifier

8 PrIMe Workflow Application

The purpose of the PrIMe Workflow Application (PWA) is to provide a user interface for working with scientific workflow projects. Using the PWA, the user can create, open, and execute scientific workflow projects. A scientific workflow project is comprised of a network of linked scientific components.

8.1 System Architecture

The general system architecture of the PWA is represented in Figure 10.

The PWA consists of the following elements:

1. *Server*. The server is the main server from which the PWA is executed. All of the functions of the server are accessed through back end core. The authorization and authentication process and the database work are accomplished by means of the core. The interaction with the client's browser and application servers is facilitated through wcf services.
2. *Application server*. Remote applications are executed remotely from the client on the application servers. Web services facilitate interaction with the client's browser and PWA.

3. *Client*. The PWA is executed from the client's browser with the use of the clickOnce application. Through the client the user can create scientific workflow projects, update existing scientific workflow projects, and execute scientific workflow projects. The clickOnce application interacts with MATLAB components through a library called ComponentsFromMatLab.dll, with which it directly communicates.
4. *MATLAB components*. The MATLAB components are downloaded to the client and activated at the execution of the scientific workflow project.

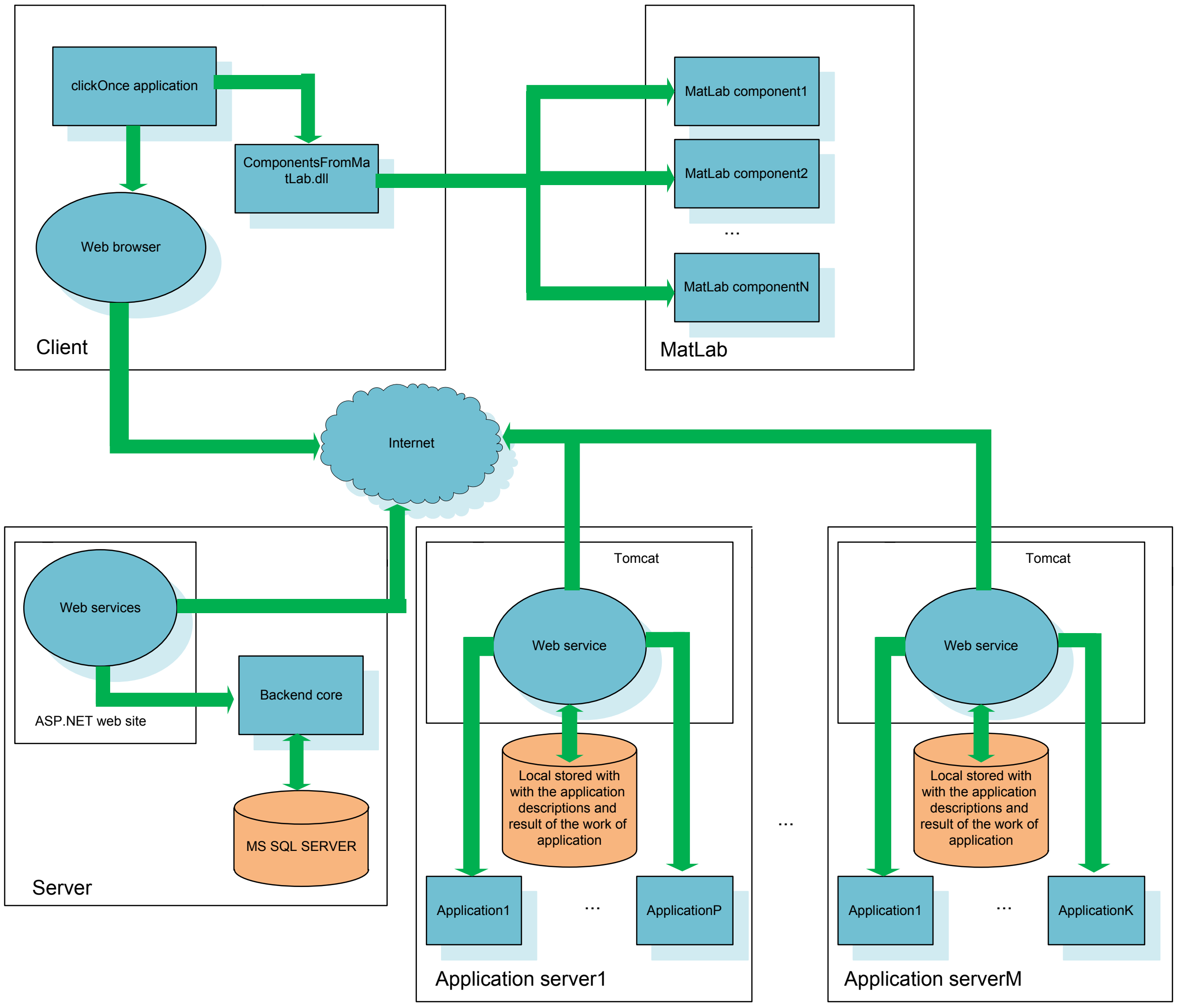


Figure 10. PWA Architecture

8.2 Use Case PWA

The function of the PWA is to work with scientific workflow projects. Use Case of the PWA is represented in Figure 11.

The main functions of the PWA are:

1. *Project management*. The PWA enables creating, editing, and deleting of scientific workflow projects.
2. *Project collaboration*. The PWA allows a user to classify a scientific workflow project as private, shared, or public. Setting a scientific workflow project as shared or public allows multiple users to collaborate on a project.
3. *Custom project building*. The PWA allows the user to create a scientific workflow project from available scientific components. The user creates the workflow project by moving scientific components to the project plane, defining the scientific component relationships with links, and specifying scientific component inputs and properties.
4. *Project execution*. Once the scientific workflow project is created, the scientific components linked, and the inputs and properties are set for each scientific component, the project can be executed in the PWA. Following execution, the results can be viewed in PWA.

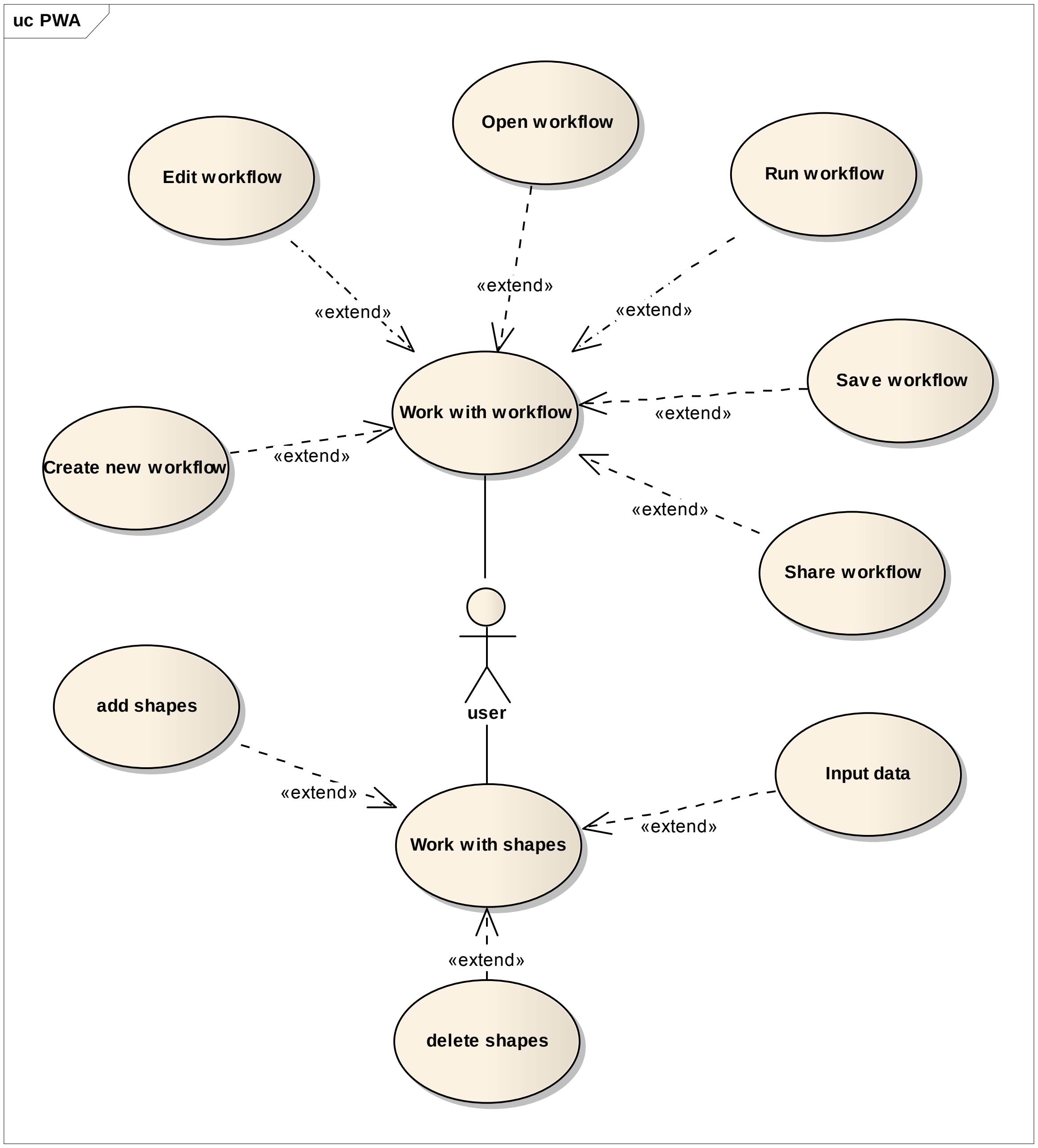


Figure 11. Use Case diagram of the PWA

8.3 Component types

There are two types of scientific components available for building of scientific workflow projects: local and remote. The following is a description of each type.

1. *Local components*. The local components created either by MATLAB or with .NET, are executed directly from the user's computer. Upon execution of a scientific

workflow project local components are copied to the client computer. The work results from each local component also are saved on the client computer.

2. *Remote components*. Remote components are executed from remote servers and interact with the PWA using web services. Multiple remote components can be hosted on a single server.

8.4 Component terms

The following terms are important in understanding components.

1. *Scientific Component (component)* – refers to a prebuilt, compiled scientific component which can be executed on a local machine or a remote server.
2. *Remote Server Application* – refers to a server application installed on remote server to manage remote component(s) execution.
3. *Shape* – a graphical representation of a scientific component. Shapes are located on PWA shape pane in a predefined shape's group. Shapes can be drag-and-dropped onto the workflow project pane.
4. *Node* – represents an instance of a shape in a workflow project. Nodes are linked into a network for workflow execution.
5. *Group* – used to visually group PWA shapes into multiple categories. Some examples of groups are: input, output, process, e.g.
6. *shapeID* – a ten character string used to uniquely identify scientific components/shape. The first five characters of shapeID are *s h a p e*. Following five characters contain five digits (i.e. shape00322). shapeID is automatically generated by the system for each scientific component.

8.5 The component integration with the PWA

8.5.1 Local components (MATLAB)

In order for the PWA to execute the local MATLAB components on a user's computer the MATLAB runtime library must be installed.

A special support library, called ComponentsFromMatLab.dll, provides the interface between the client browser and the local scientific components. clickOnce application communicates with the local scientific components through the ComponentsFromMatLab.dll library. Upon execution of a scientific workflow project, each component is executed by means of the library methods that control it. At the beginning of project execution, the support library downloads each MATLAB component to the local computer. Each component is executed by the MATLAB runtime library. The support library, ComponentsFromMatLab.dll, controls the execution of each local scientific component.

When the scientific workflow project is executed, all of the project information is stored on the client's computer in a catalog called ProjectData.xml. This catalog encodes the scientific component relationships, description, properties, the location of results, and the completion status of each component. The paths of each component results are specified in the catalog.

ProjectData.xml is recorded only once at the time of scientific workflow project execution. Below, an xml example is represented. Each component in the project is represented by a <node> element, in which all of the input properties and information about the connected components are recorded. The completion status of each component is recorded as 1 (success) or 0 (failure).

```
<?xml version="1.0" encoding="utf-8" ?>
<project id="165" modified="29.09.2008 15:26:55" executed="" creator="alx" status="1" lastInternalId="8" >
  <nodes>
    <node id="2" name="Model 1" group="Models" type="Model" >
      <properties>
        <status>0</status>
        <resultObj></resultObj>
        <about>This node supplies a model.</about>
        <icon>model.gif</icon>
        <list description="Model Source" group="attributes" name="Source" readOnly="False">
          <option value="1" caption="PrIMe Warehouse" link="http://prime-
          warehouse.berkeley.edu/depository/models/catalog/m00000003.xml" selected="true" >from PrIMe Warehouse</option>
          <option value="2" caption="local" link="" >from local machine</option>
        </list>
        <list description="Model Type" group="attributes" name="Type" readOnly="False">
          <option value="1" caption="detailed" link="" selected="true" >A detailed model</option>
          <option value="2" caption="reduced" link="" >reduced model</option>
          <option value="3" caption="tabulated" link="" >tabulated model</option>
        </list>
      </properties>
      <location width="92" height="62" x="43" y="315" />
      <layout>
        <image x="0" y="0" file="model.gif" />
      </layout>
      <inputs>
      </inputs>
      <outputs>
        <output id="1" x="45" y="54" linkNodeId="4"
         linkNodeInputId="1">c:/PrIMe_Workflow/projects/project_165/nodes/node_2/grimech30.mat</output>
      </outputs>
    </node>
  </nodes>
</project>
```

We now review the ComponentFromMatLab.dll in more detail. The purpose of the ComponentsFromMatLab.dll library is to provide methods for scientific components developed by third-party developers, such as custom built MATLAB programs, to be integrated with the PWA. Scientific components must follow integration rules defined in the PWA (ProjectData.xml). The interface library must contain one or more classes, with methods having the following prototype:

```
public static string outputXml RunComponent(string inputXml)
```

Where inputXML contains input information required for component execution, and outputXML contains the component's execution results.

Input XML

This section represents sample input XML and describes input XML elements.

```
<inputData>
	<currentNode shapeID="shape00105">
		<outputDir value="C:\Prime_Workflow\projects\project_1\nodes\node_1"/>
		<userInput>
			<float description="Reactor residence time (msec)" group="attributes" name="Residence time
(msec)" readOnly="False" value="0,233"/>
			<list description="Code Specification" group="attributes" name="Numerical Code" readOnly="False">
```

```
                    <option value="1" caption="Matlab application"
link="http://reactionlab.sourceforge.net/">ReactionLab</option>
                    <option value="2" caption="Code of Heinz Pitsch" link=""
selected="true">FlameMaster</option>
                    <option value="3" caption="" link="">Other</option>
                </list>
            </userInput>
        </currentNode>
        <linkedNodes>
            <linkedNode shapeID="shape00108" >
                <userInput>
                    <float description="Pressure" group="Init state" name="Pressure" readOnly="False"
value="12"/>
                </userInput>
                <resultFiles>
                    <file>C:\Prime_Workflow\projects\project_1\nodes\node_2\result1.data</file>
                    <file>C:\Prime_Workflow\projects\project_1\nodes\node_2\result2.data</file>
                </resultFiles>
            </linkedNode>
            <linkedNode shapeID="shape00109">
                <userInput>
                    <bool description="Temperature" group="Init state" name="Bool value" readOnly="True"
value="False"/>
                </userInput>
                <resultFiles>
                    <file>C:\Prime_Workflow\projects\project_1\nodes\node_3\result.data</file>
                </resultFiles>
            </linkedNode>
        </linkedNodes>
</inputData>
```

The input XML's root node inputData has two child elements: currentNode and linkedNodes. CurrentNode element contains output directory and user input information for the currently executed node.

outputDir – specifies where the current component should deposit result data files.

userInput – provides component properties and other information entered by user.

linkNodes element contains information about all input nodes (currently we support up to two input nodes).

LinkNode is a child of linkNodes element. It contains userInput data, resultFiles location, and other node properties.

Output XML

After component execution is completed, the component should return a string in XML format (outputXML). The outputXML contains the execution status and other component-specific information. At this point, additional information is the pft-type, error text (if status=2), which is necessary to change in the diagram after the component completed. We can expand the list of return values, by adding new sections in the outputData XML. The status field can have the following values:

Value	**Description**
1	Success
2	Failed

In any case, the component should return a value. Until then the system is blocked, and will not be able to run other components.

Below is an example of the component output xml:

```
<outputData>
      <status> 2 </ status>
      <errorMessage>message text</errorMessage>
      <nodeCaption>Model7.1</ nodeCaption>
      <nodeProperties>some data</ nodeProperties >
</ outputData>
```

Below is an example of a .NET interface for local MATLAB components:

```
using System;
using System.Collections.Generic;
using System.Text;
public class MatlabCompsSupportClass
{
   public static string RunPFR(string input)
   {
      MatlabComponents mc = new MatlabComponents(@"c:\PrIMe_Workflow\matlab_comps");// instantiating main class for
//components bundle
      return mc.run_pfr(input).ToString();// running the component
   }
}
```

It is necessary to modify the main stub class generated by MATLAB Builder for .NET in order to pass the CTF file location explicitly; otherwise the current directory is used by default. The only requirement is to replace the static constructor with a constructor having the installation path as a parameter. An example is shown below.

```
public MatlabComponents(string ctfFilePath)
   {
      if (MWArray.MCRAppInitialized && mcr == null)
      {
         mcr = new MWMCR(MCRComponentState.MCC_matlab_comps_name_data,
                     MCRComponentState.MCC_matlab_comps_root_data,
                     MCRComponentState.MCC_matlab_comps_public_data,
                     MCRComponentState.MCC_matlab_comps_session_data,
                     MCRComponentState.MCC_matlab_comps_matlabpath_data,
                     MCRComponentState.MCC_matlab_comps_classpath_data,
                     MCRComponentState.MCC_matlab_comps_libpath_data,
                     MCRComponentState.MCC_matlab_comps_mcr_application_options,
                     MCRComponentState.MCC_matlab_comps_mcr_runtime_options,
                     MCRComponentState.MCC_matlab_comps_mcr_pref_dir,
                     MCRComponentState.MCC_matlab_comps_set_warning_state,
                     ctfFilePath, true); // pass contuctor parameter to MWCR initializer
      }
      else
      {
         throw new ApplicationException("MWArray assembly could not be initialized");
      }
   }
```

8.5.2 Local components (.NET)

As was mentioned above, PWA supports two types of local components; components created in MATLAB and components created in .NET. The .NET component interface is identical to MATLAB components. For more details see sample Input and Output XML above in section “local components (MATLAB)”. The difference with .NET components is that you don’t need to compile and install MCR, the MATLAB runtime library, and you don't need to compile MatlabCompsSupportClass.

8.5.3 Remote components

Remote server application is implemented in Java to support multiplatform implementation requirement.

Remote components are configured in PWA Uploader application. Among other information, the component developer should provide remote component (application) name, application id, remote server IP address and port number to be used. In addition, the component developer may configure the user's work group association. This information is used to limit component execution access only to authorized users.

Each remote component execution request (job) is assigned a unique jobid. The following job attributes are logged during the remote component execution: projectid, nodeid, applicationid, execution status, and jobid.

PWA remote components are executed as a part of PWA project. Here is what happens during the execution:

1. PWA client execution process makes request to the PWA server to check component execution status.
2. Next PWA client validates if user has permissions to run the remote component.
3. If component is not running the PWA client makes a web service call to the Remote Server Application to launch the remote component.
4. When remote component is launched, it's responsible for creating a status file and setting status file to 0 – "processing"
5. PWA client starts polling remote component execution status.
6. The PWA user can cancel remote component execution at any point by pressing a cancel button.
7. When remote component completes execution it populates all fields in the status file and sets status to 1-success or 2-failed. If execution failed, the component has to populate an error message into status file.
8. At this point execution is returned back to PWA client.

Input parameters

Remote components are launched by remote server application from command line with one XML input parameter. The XML parameter contains information about input nodes, user entered input and user information (e.g. userid, groupid, user login).

Below is demonstrated a sample input XML and description of input XML elements.

```
<inputData>
```

```
    <userInfo>
        <userId>330</userId>
        <login>aljokan</login>
        <groups>
            <group>PrIMe Team</group>
            <group>ReactionDesign</group>
        </groups>
    </userInfo>
    <currentNode shapeID="shape0118">
        <outputDir value="C:\prime\jobs\job_270\nodes\node_3" />
        <userInput>
            <float description="Residence Time in seconds." group="attributes" name="Residence Time (sec)" readOnly="False"
value="1" />
            <list description="System Heat Constrains" group="attributes" name="Energy Control" readOnly="False">
                <option value="1" caption="adiabatic" link="" selected="true">No heat exchange</option>
                <option value="2" caption="isothermal" link="">Constant temperature</option>
            </list>
            <list description="Process" group="attributes" name="Process" readOnly="False">
                <option value="1" caption="isobaric" link="" selected="true">Constant pressure</option>
                <option value="2" caption="isochoric" link="">Constant volume</option>
            </list>
        </userInput>
    </currentNode>
    <linkedNodes>
        <linkedNode shapeID="shape00103">
            <userInput>
                <list description="Target Source" group="attributes" name="Source" readOnly="False">
                    <option value="1" caption="PrIMe Warehouse" link="" selected="true">from PrIMe Warehouse</option>
                    <option value="2" caption="local" link="">from local machine</option>
                </list>
            </userInput>
            <resultFiles />
        </linkedNode>
        <linkedNode shapeID="shape00107">
            <userInput>
                <list description="Model Source" group="attributes" name="Source" readOnly="False">
                    <option value="1" caption="PrIMe Warehouse" link="http://prime-
warehouse.berkeley.edu/depository/models/catalog/m00000003.xml" selected="true">from PrIMe Warehouse</option>
                    <option value="2" caption="local" link="">from local machine</option>
                </list>
            </userInput>
            <resultFiles>
                <file>C:\prime\jobs\job_270\nodes\node_2\grimech30.h5</file>
            </resultFiles>
        </linkedNode>
    </linkedNodes>
</inputData>
```

inputData is a root node with following child elements:

currentNode, linkedNodes, userInfo and groups

userInfo and groups elements provide information about the user which performs the request, and user’s work group association.

currentNode element contains node unique id, output directory and user input information for currently executed node.

outputDir – specifies where current component should deposit resulted data files

userInput – provides component properties and other information entered by user

linkNodes element contains information about all input nodes (currently we support up to two input nodes.

LinkNode is a chiled of linkNodes element. It contains userInput data, resultFiles location, and other node properties.

Below is an example how a remote server application launches a remote component from the command line:

../prime/components/component1 [illegible]*sr/local/prime/projects/project_1/data.xml*

Output parameters

When a remote component starts execution, it should create a file (in component’s folder) called 'status' and set status value to 0. The file will contain execution state code and error message. The status can have one of the following values:

Value	**Description**
0	Processing
1	Success
2	Failed

Status File

Below is an example of output status file:

```
<outputData>
    <status>2</status>
    <errorMessage>
    ERROR: Reading Reactor Model Inputs: Did not find InitialState file; unable to initialize reactor.
    This file is produced by connecting the 'State' input object to the reactor object.
    ERROR: Reading Reactor Model Inputs: Unable to obtain required input files or output file location.
    Please rerun your project from the beginning.
    </errorMessage>
</outputData>
```

8.6 Executing a Project

After creating the scientific workflow project, the user can execute it. The scientific workflow can consist either of local components (MATLAB) or of remote components, which are implemented on remote application servers. The execution logic is the same for remote and local components. In Figure 12 the process of executing a scientific workflow project is shown.

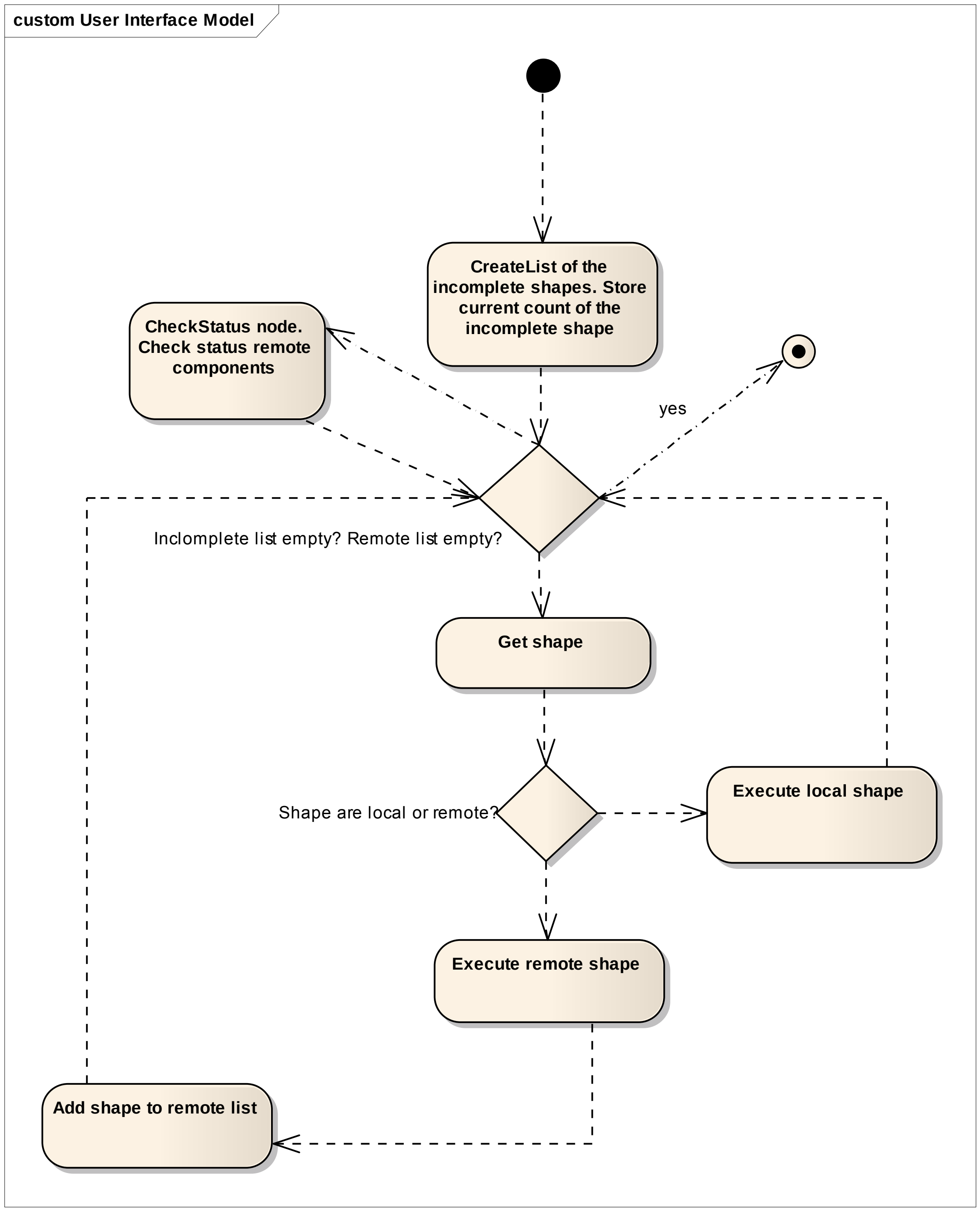

custom User Interface Model
CreateList of the incomplete shapes. Store current count of the incomplete shape
CheckStatus node. Check status remote components
yes
Inclomplete list empty? Remote list empty?
Get shape
Shape are local or remote?
Execute local shape
Execute remote shape
Add shape to remote list

Figure 12. Activity diagram of executing a scientific workflow project

The process to start the execution of local components is trivial. The appropriate method from the support library simply is activated. The execution of a remote component is represented more exactly in Figures 13 and 14.

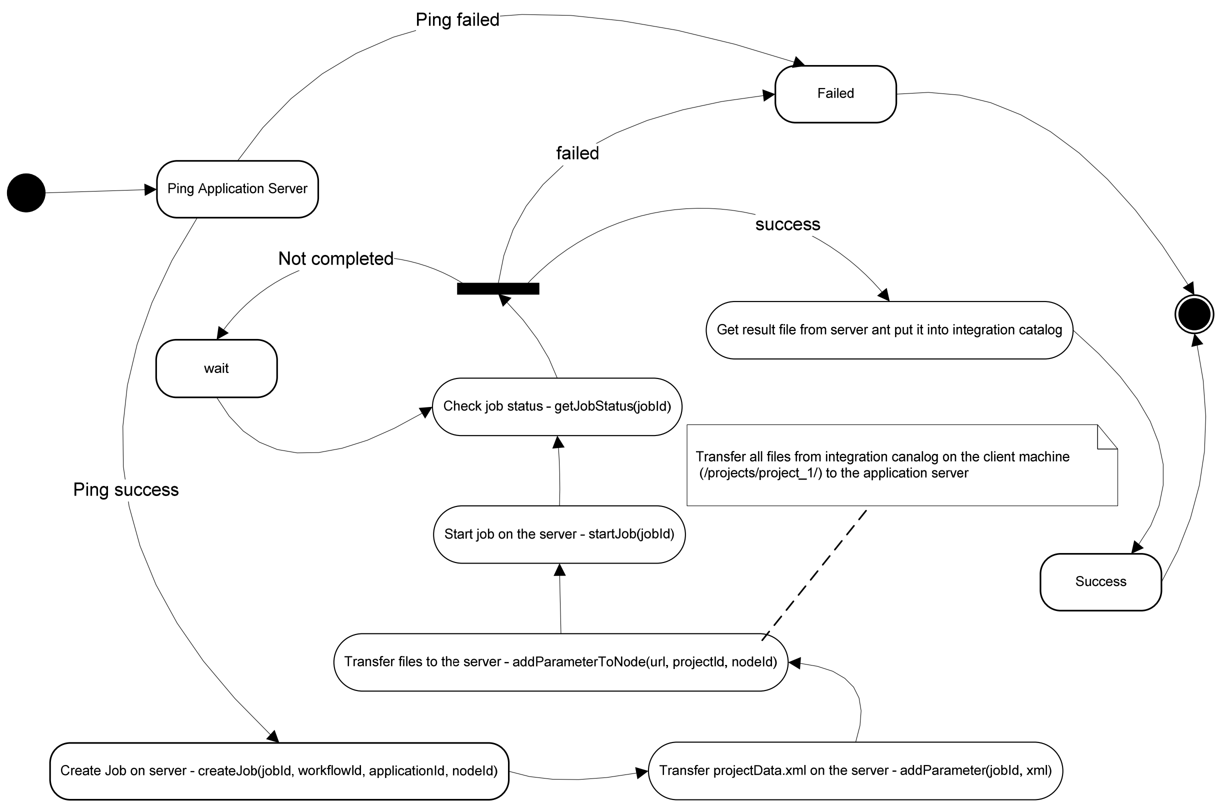

Ping failed
Failed
failed
Ping Application Server
success
Not completed
Get result file from server ant put it into integration catalog
wait
Check job status - getJobStatus(jobId)
Transfer all files from integration canalog on the client machine (/projects/project_1/) to the application server
Ping success
Start job on the server - startJob(jobId)
Success
Transfer files to the server - addParameterToNode(url, projectId, nodeId)
Create Job on server - createJob(jobId, workflowId, applicationId, nodeId)
Transfer projectData.xml on the server - addParameter(jobId, xml)

Figure 13. State diagram of execution of the remote component

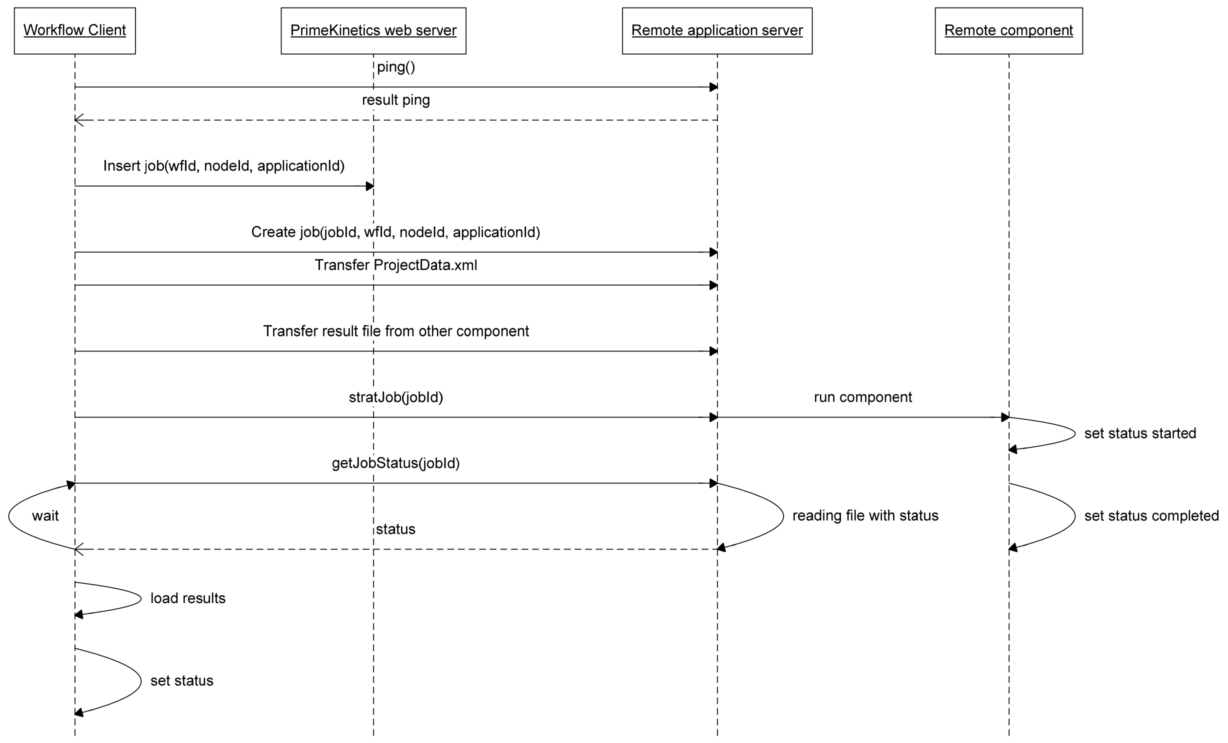


Figure 14. Activity diagram of executing a remote component

8.7 User's computer

Upon starting the PWA, the following three libraries are copied to the client computer: PrimMeKineticsClient.dll, ComponentsFromMatLab.dll, and matlab_component.dll. All functions for working with diagrams, starting project implementation, and interaction with server applications are executed in these libraries. Figure 15 shows all of the elements of the PWA that are stored on the client computer.

PrIMeKineticsClient.dll—This library is implemented as clickOnce application, in which all the functions for creating and editing scientific workflow projects and their execution are implemented.

Support Library (ComponentFromMatLab.dll)—This library represents the interface for the interaction with MATLAB components. PrIMeKineticsClient.dll does not know about the components and their structure, yet it only knows the methods of the support library by which they are controlled.

MATLAB component (matlab_comps.dll)—This is the library that directly controls the programs from MATLAB. This library is generated by the MATLAB Builder for the .NET application. The MATLAB component library is connected to the support library and represents the required classes and functions for MATLAB programs.

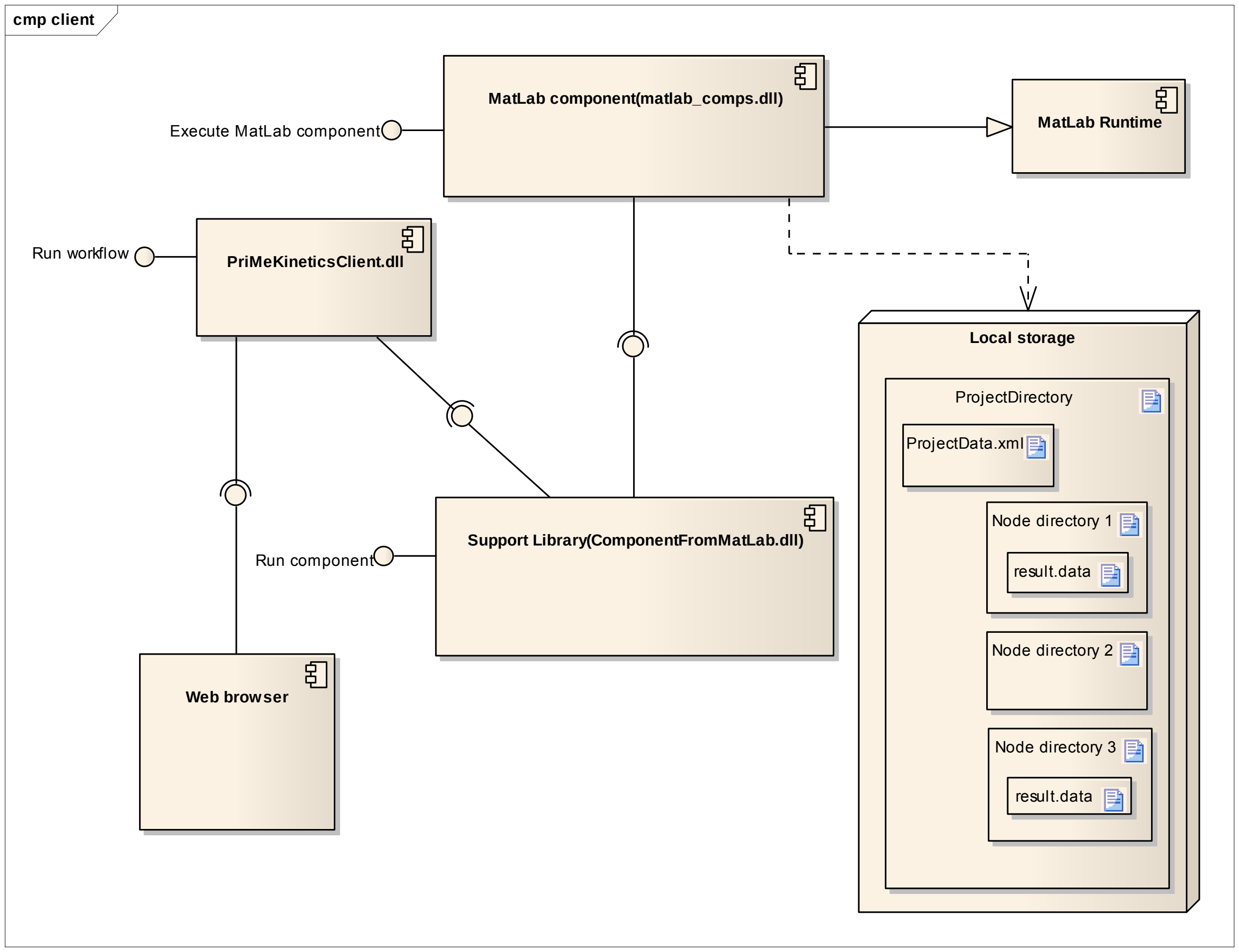


Figure 15. System components which are located on the user's computer

8.7.1 Main modules of the PrIMeKineticsClient.dll library

The general structure of the PrIMeKineticsClient.dll library is represented in Figure 16. It consists of the following parts:

1. *Workflows*. This module provides the methods that control scientific workflow projects such as creation, opening, and deletion of projects and managing of the system users' access to the existing projects.
2. *Shapes*. In this module the graphical editor is controlled. It includes methods that control how a component is displayed, the properties and input data of each component, and the relationships of each component in a scientific workflow project.

3. *Execute workflow*. This module controls how each component is executed and how the scientific workflow project is executed as a whole. The module manages the process and order of execution of every component and identifies whether the component is local or remote.

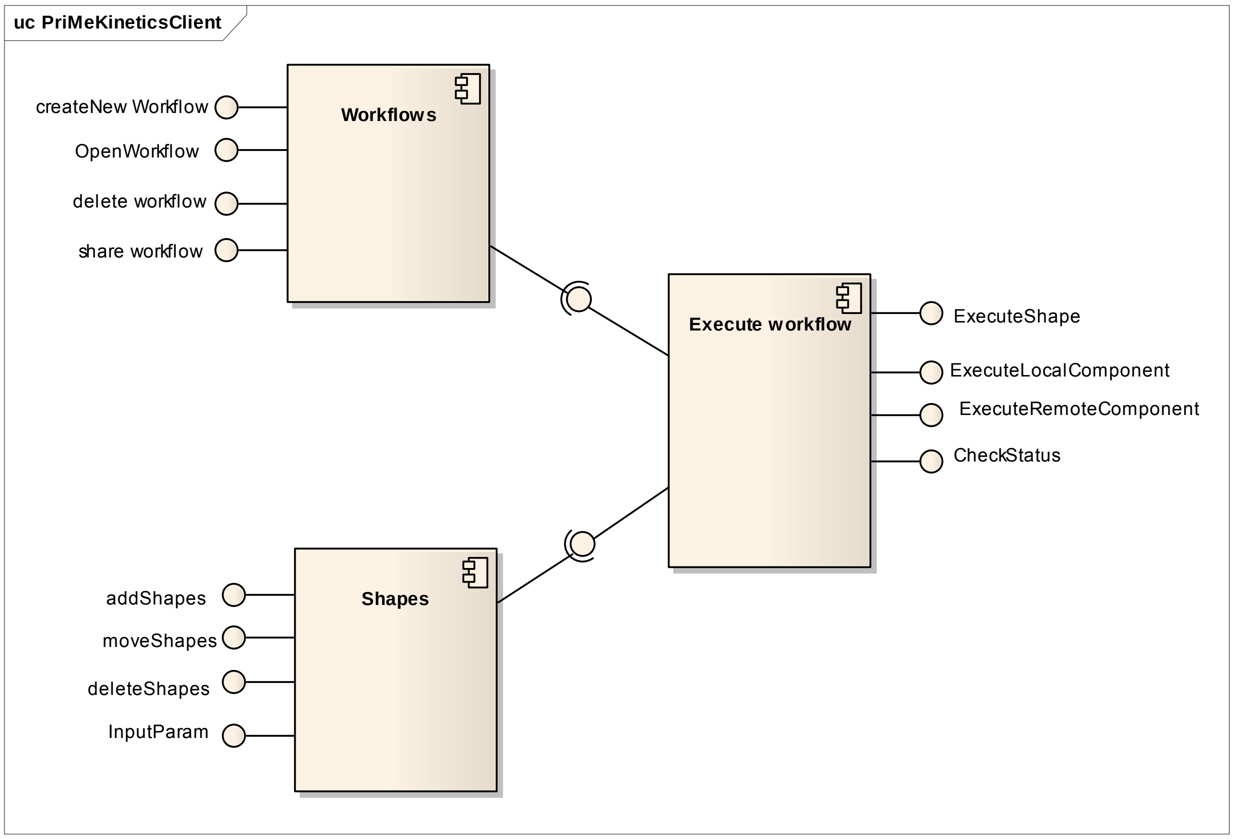


Figure 16. General PrIMeKineticsClient.dll structure

8.7.2 Classes diagram

The Classes diagrams for the PrIMeKineticsClient.dll library are presented in Figures 17 and 18. Below a description of each method and class assignment is presented.

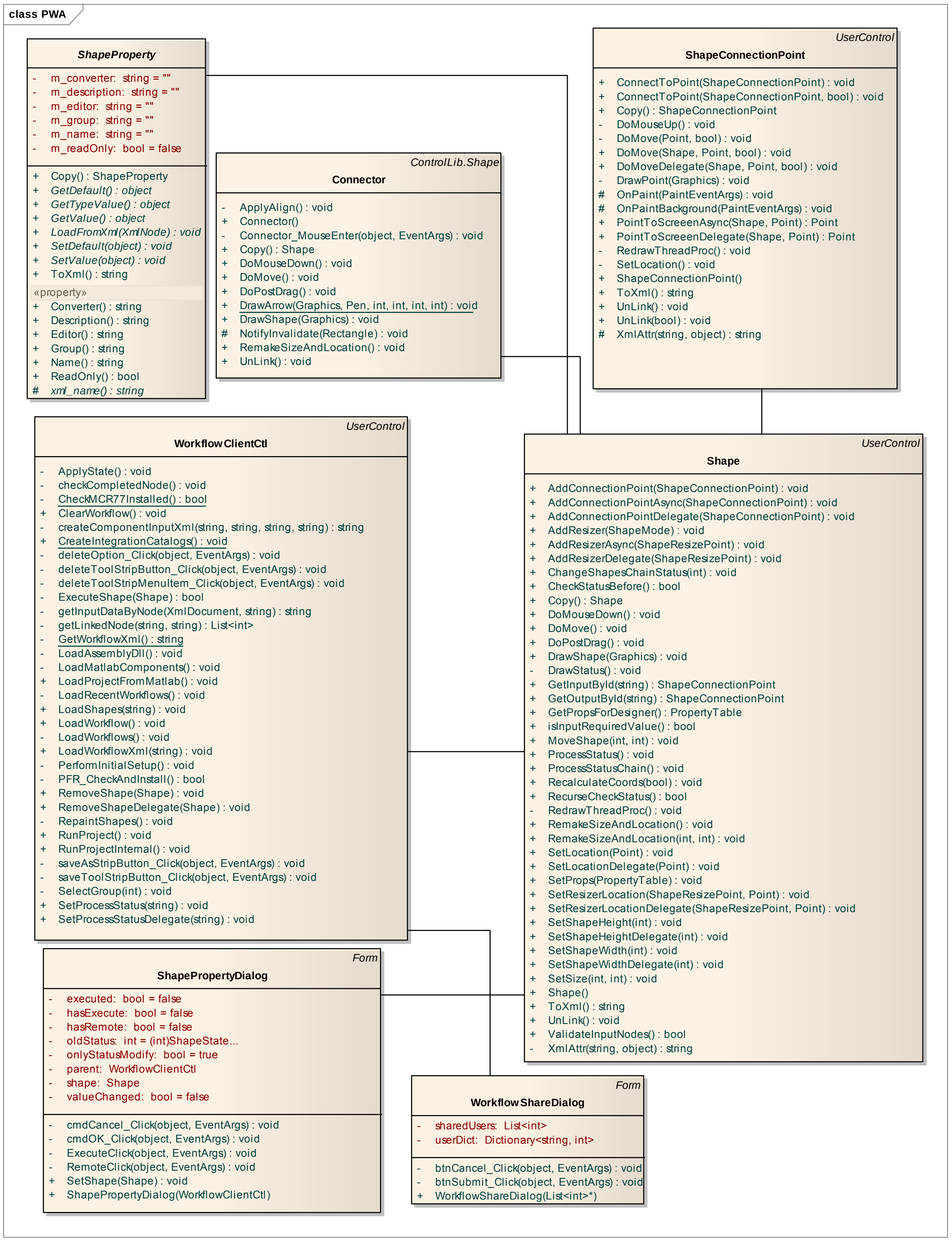

class PWA
ShapeProperty
- m_converter: string = ""
- m_description: string = ""
- m_editor: string = ""
- m_group: string = ""
- m_name: string = ""
- m_readOnly: bool = false
+ Copy() : ShapeProperty
+ GetDefault() : object
+ GetTypeValue() : object
+ GetValue() : object
+ LoadFromXml(XmlNode) : void
+ SetDefault(object) : void
+ SetValue(object) : void
+ ToXml() : string
«property»
+ Converter() : string
+ Description() : string
+ Editor() : string
+ Group() : string
+ Name() : string
+ ReadOnly() : bool
# xml_name() : string
ControlLib.Shape
Connector
- ApplyAlign() : void
+ Connector()
- Connector_MouseEnter(object, EventArgs) : void
+ Copy() : Shape
+ DoMouseDown() : void
+ DoMove() : void
+ DoPostDrag() : void
+ DrawArrow(Graphics, Pen, int, int, int, int) : void
+ DrawShape(Graphics) : void
# NotifyInvalidate(Rectangle) : void
+ RemakeSizeAndLocation() : void
+ UnLink() : void
UserControl
ShapeConnectionPoint
+ ConnectToPoint(ShapeConnectionPoint) : void
+ ConnectToPoint(ShapeConnectionPoint, bool) : void
+ Copy() : ShapeConnectionPoint
- DoMouseUp() : void
- DoMove(Point, bool) : void
+ DoMove(Shape, Point, bool) : void
+ DoMoveDelegate(Shape, Point, bool) : void
- DrawPoint(Graphics) : void
# OnPaint(PaintEventArgs) : void
# OnPaintBackground(PaintEventArgs) : void
+ PointToScreeenAsync(Shape, Point) : Point
+ PointToScreeenDelegate(Shape, Point) : Point
- RedrawThreadProc() : void
- SetLocation() : void
+ ShapeConnectionPoint()
+ ToXml() : string
+ UnLink() : void
+ UnLink(bool) : void
# XmlAttr(string, object) : string
UserControl
WorkflowClientCtl
- ApplyState() : void
- checkCompletedNode() : void
- CheckMCR77Installed() : bool
+ ClearWorkflow() : void
- createComponentInputXml(string, string, string, string) : string
+ CreateIntegrationCatalogs() : void
- deleteOption_Click(object, EventArgs) : void
- deleteToolStripButton_Click(object, EventArgs) : void
- deleteToolStripMenuItem_Click(object, EventArgs) : void
- ExecuteShape(Shape) : bool
- getInputDataByNode(XmlDocument, string) : string
- getLinkedNode(string, string) : List<int>
- GetWorkflowXml() : string
- LoadAssemblyDll() : void
- LoadMatlabComponents() : void
+ LoadProjectFromMatlab() : void
- LoadRecentWorkflows() : void
+ LoadShapes(string) : void
+ LoadWorkflow() : void
- LoadWorkflows() : void
+ LoadWorkflowXml(string) : void
- PerformInitialSetup() : void
- PFR_CheckAndInstall() : bool
+ RemoveShape(Shape) : void
+ RemoveShapeDelegate(Shape) : void
- RepaintShapes() : void
+ RunProject() : void
+ RunProjectInternal() : void
- saveAsStripButton_Click(object, EventArgs) : void
- saveToolStripButton_Click(object, EventArgs) : void
- SelectGroup(int) : void
+ SetProcessStatus(string) : void
+ SetProcessStatusDelegate(string) : void
UserControl
Shape
+ AddConnectionPoint(ShapeConnectionPoint) : void
+ AddConnectionPointAsync(ShapeConnectionPoint) : void
+ AddConnectionPointDelegate(ShapeConnectionPoint) : void
+ AddResizer(ShapeMode) : void
+ AddResizerAsync(ShapeResizePoint) : void
+ AddResizerDelegate(ShapeResizePoint) : void
+ ChangeShapesChainStatus(int) : void
+ CheckStatusBefore() : bool
+ Copy() : Shape
+ DoMouseDown() : void
+ DoMove() : void
+ DoPostDrag() : void
+ DrawShape(Graphics) : void
- DrawStatus() : void
+ GetInputById(string) : ShapeConnectionPoint
+ GetOutputById(string) : ShapeConnectionPoint
+ GetPropsForDesigner() : PropertyTable
+ isInputRequiredValue() : bool
+ MoveShape(int, int) : void
+ ProcessStatus() : void
+ ProcessStatusChain() : void
+ RecalculateCoords(bool) : void
+ RecurseCheckStatus() : bool
- RedrawThreadProc() : void
+ RemakeSizeAndLocation() : void
+ RemakeSizeAndLocation(int, int) : void
+ SetLocation(Point) : void
+ SetLocationDelegate(Point) : void
+ SetProps(PropertyTable) : void
+ SetResizerLocation(ShapeResizePoint, Point) : void
+ SetResizerLocationDelegate(ShapeResizePoint, Point) : void
+ SetShapeHeight(int) : void
+ SetShapeHeightDelegate(int) : void
+ SetShapeWidth(int) : void
+ SetShapeWidthDelegate(int) : void
+ SetSize(int, int) : void
+ Shape()
+ ToXml() : string
+ UnLink() : void
+ ValidateInputNodes() : bool
- XmlAttr(string, object) : string
Form
ShapePropertyDialog
- executed: bool = false
- hasExecute: bool = false
- hasRemote: bool = false
- oldStatus: int = (int)ShapeState...
- onlyStatusModify: bool = true
- parent: WorkflowClientCtl
- shape: Shape
- valueChanged: bool = false
- cmdCancel_Click(object, EventArgs) : void
- cmdOK_Click(object, EventArgs) : void
- ExecuteClick(object, EventArgs) : void
- RemoteClick(object, EventArgs) : void
+ SetShape(Shape) : void
+ ShapePropertyDialog(WorkflowClientCtl)
Form
WorkflowShareDialog
- sharedUsers: List<int>
- userDict: Dictionary<string, int>
- btnCancel_Click(object, EventArgs) : void
- btnSubmit_Click(object, EventArgs) : void
+ WorkflowShareDialog(List<int>*)

Figure 17. PrIMeKineticsClient.dll classes diagram

Shape—The components element, its presentation, setting properties and editing are controlled by this class

Method	**Description**
AddConnectionPoint	Adds entry or exit to shape
ChangeShapesChainStatus	Changes the shape status at editing its connections with other shapes.
Copy	Creates a copy of the shape. Is activated when the user drags a new shape on the diagram
DoMouseDown	Processes the user's mouse clicking on shape
isInputRequiredValue	Checks whether or not the field is required for execution
MoveShape	Processes the relocation of shapes
SetLocation	Positions the shapes in a specified location
SetProps	Sets properties that are entered by the user
ToXml	Converts the shape and all its properties to xml
UnLink	Activates a connection deleting connections with other elements on a diagram
ValidateInputNodes	Validates inputs and outputs
XmlAttr	Returns by name the attribute value
GetInputById	Returns the shapes input by identifier
GetOutputById	Returns shapes output by identifier
SetSize	Sets shape size
RedrawThreadProc	Displays a semi-transparent flow diagram when moving shapes.

ShapeConnectionPoint—This class used for the connecting components displayed in the scientific workflow project

Method	**Description**
ConnectToPoint	Connects the set point with another set point
SetLocation	Sets the position of the shape
DrawPoint	Displays a point on the screen
ToXml	Converts all the point properties to xml
UnLink	Deletes the connection of the selected point

Connector—Is used for displaying lines which connect two diagram elements

Method	**Description**
doMouseDown	Processes the user left mouse click and starts to draw the connecting line
doMove	Displays the connector while the user moves it around the diagram
remakeSizeAndLocation	Changes the size and position of the connector while it is being moved.
UnLink	Removes the connector
ApplyAlign	Applies the changes set by user

ShapeProperty—Is used to set shape properties

Method	**Description**
LoadFromXml	Downloads property from xml
SetValue	Sets property value
ToXml	Converts property to xml
GetValue	Returns the property value

WorkFlowClientCtl—Used by main GUI library

Method	**Description**
ApplyState	Applies a new condition to the diagram
CheckCompletedNode	Checks diagram nodes
CheckMCR77Installes	Checks whether or not the MATLAB Runtime is installed
ClearWorkflows	Creates a new workflow
ExecuteShape	Executes the component in the application, which is connected with the indicated shape
getInputDataByNode	Returns the shape properties, inputted by the user
LoadAssemblyDll	Downloads client component
LoadProjectFromMatLab	Downloads xml which MATLAB modified when the component was executed

LoadWorkflow	Opens workflow
RunProject	Executes the project at the users request
RepaintShapes	Repaints shapes on the diagram
RunProjectInternal	Runs the project in separate flow and manages the diagram starting process
SetProcessStatus	Displays the diagram starting progress
SaveAsStripButton_Click	Stores the project
getLinkedNode	Returns all the shapes that are connected on the diagram with the set
CreateIntagrationCatalog	Creates the catalogue on the user's computer, to which work results will be stored
PerformInitialSetup	Activates the workflow after downloading, makes all the necessary initialization, and displays the library download process in client's browser
RemoveShapes	Deletes the shape from the diagram
LoadRecentWorkflows	Downloads recent workflows that are available

WorkflowShareDialog—GUI to provide the user access to shared projects

Method	**Description**
btnSubmitClick	Applies rights set by the user
btnCancel_Click	Processes when the user clicks cancel
workflowShareDialog	Downloads and displays the list of all the system users with which the workflow is shared

ShapePropertyDialog—GUI that sets the shape property

Method	**Description**
cmdCancel_click	Processes when the user clicks cancel
cmdOK_Click	Applies all the properties set by the user
SetShape	Connects the selected shape from the GUI data, displays the inputted properties, and remembers the shape for storing new properties

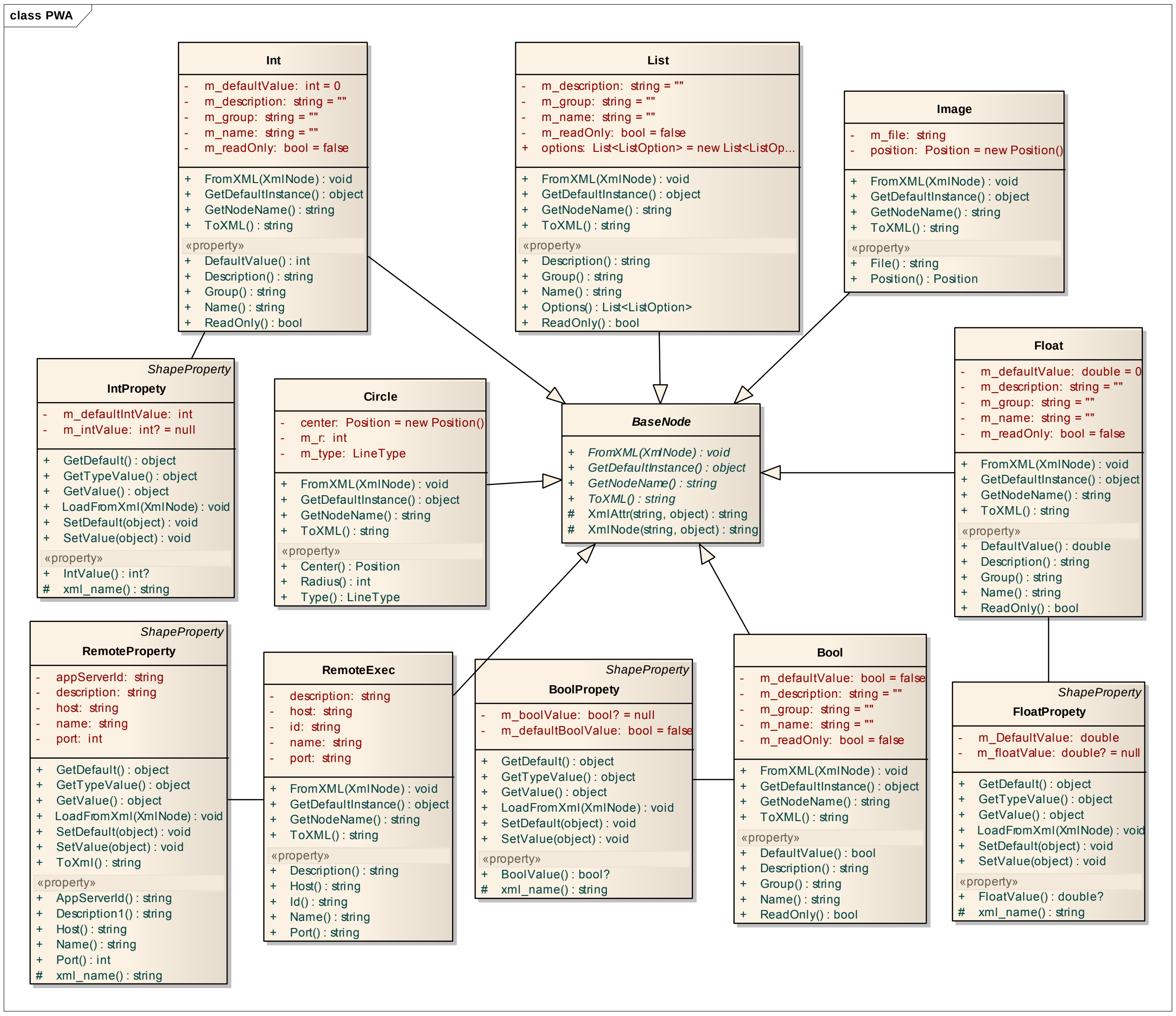


Figure 18. PrIMeKineticsClient.dll class diagram

Classes displayed in Figure 18 are used for the storing information about the scientific workflow projects and their structure stored in ProjectData.xml.

BaseNode—The base class which is used for storing information in ProjectData.xml

Method	**Description**
FromXml	Creates the node on the basis of the xml description
GetNodeName	Returns the node name
XmlAttr	Returns the attribute by indicated name
XmlNode	Creates XmlNode from indicated information

All the other classes Bool, Int, Float, List, RemoteExec—inherit from the BaseNode class. Their methods are trivial and hence their description is omitted.

8.8 Application server structure

The Application server stores the applications that are implemented remotely from the client's computer. The interaction with the application server is accomplished through web services. The main components of the application server are discussed below.

Web service—Web service is installed on the application server. Web service provides the interface for the interaction with PWA. Web service also allows input parameters, component properties, and component results to be communicated between the application server and the main server.

Application—The application makes the necessary calculations based on the input parameters. Each application must correspond to defined requirements, which are described in the following paragraph.

Local storage (config files) —These are the necessary configuration files, used by the web application.

1. *Application.xml*—A file where the applications register and the path of the execution file is indicated.
2. *Jobs.property*—A property file where the information about the current tasks is stored.
3. *Config.properties*—A configuration profile where the indicated path to the catalog of application results is saved. This file also creates a log and stores the path of the application.xml and jobs.property files.
4. *ProjectDirectory*—For each project a project directory is created that stores the project information in a ProjectData.xml file. Each project node has its own directory where the specific node results are stored.

8.8.1 Application server structure

In Figures 19 and 20 the structure and modules of the Application server are shown. The web service facilitates the interaction of the application server with the PWA and manages the application starting process.

The main modules are presented below.

1. *WebService*. This module calls the methods which are used for the interaction with PWA, and manages the process of starting the application on the Application server. ComponentRun. This module also controls the execution of the scientific application.
2. *Config*. This module provides access to the main configuration files.

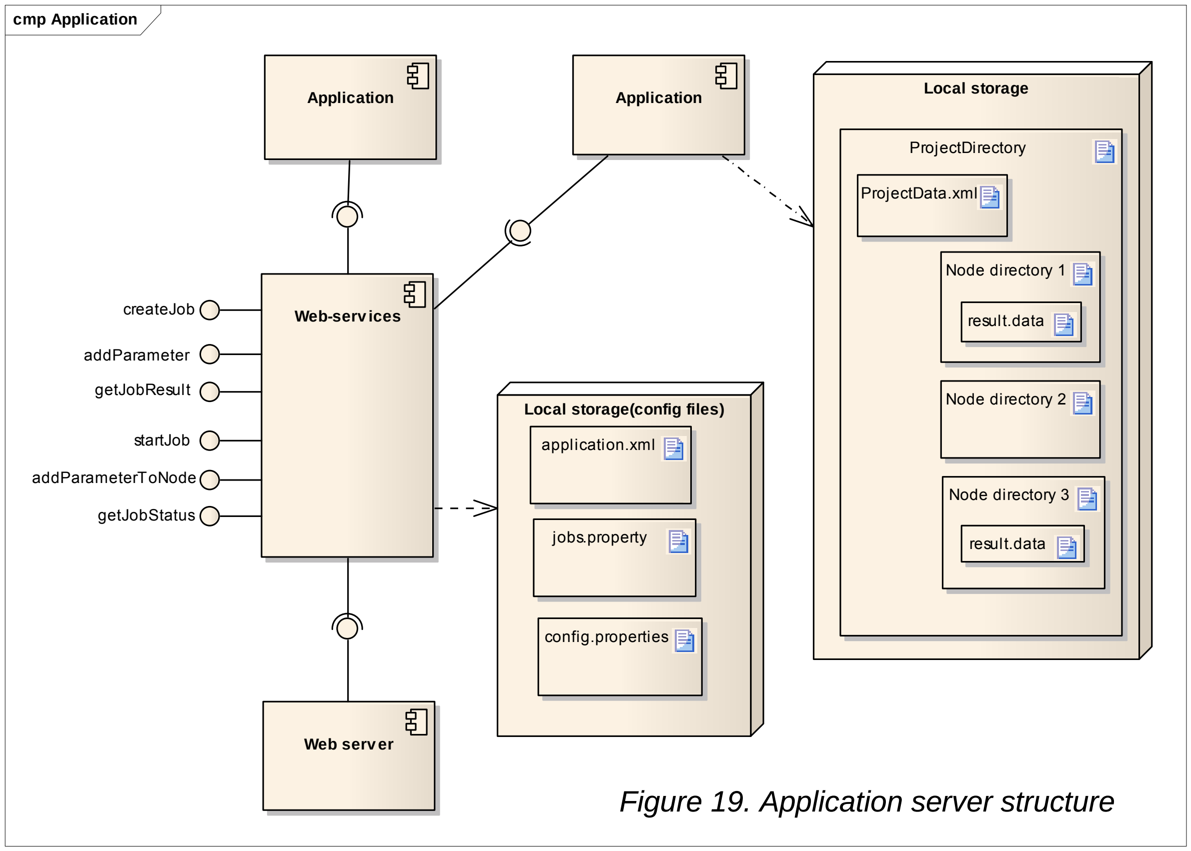


Figure 19. Application server structure

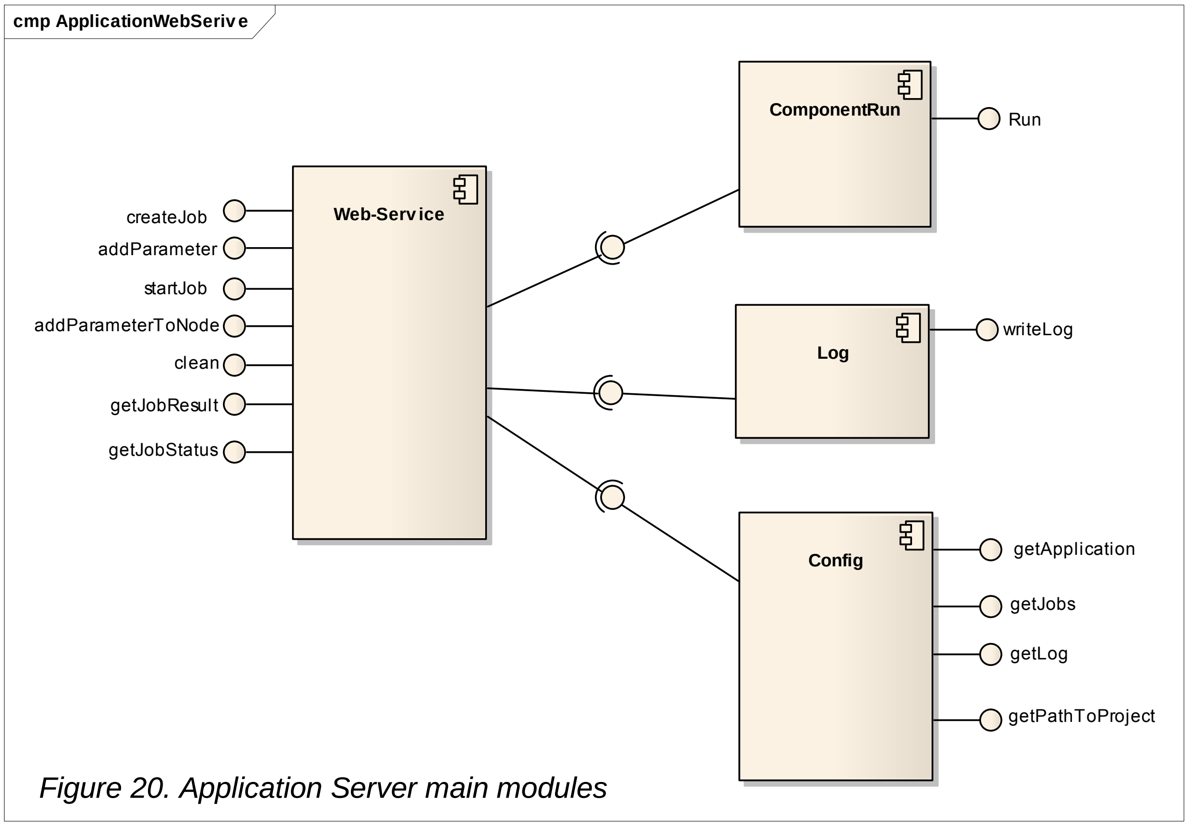


Figure 20. Application Server main modules

8.8.2 Classes diagram

The main classes of the Application server are represented in Figure 21. A description of the Application server and classes follows.

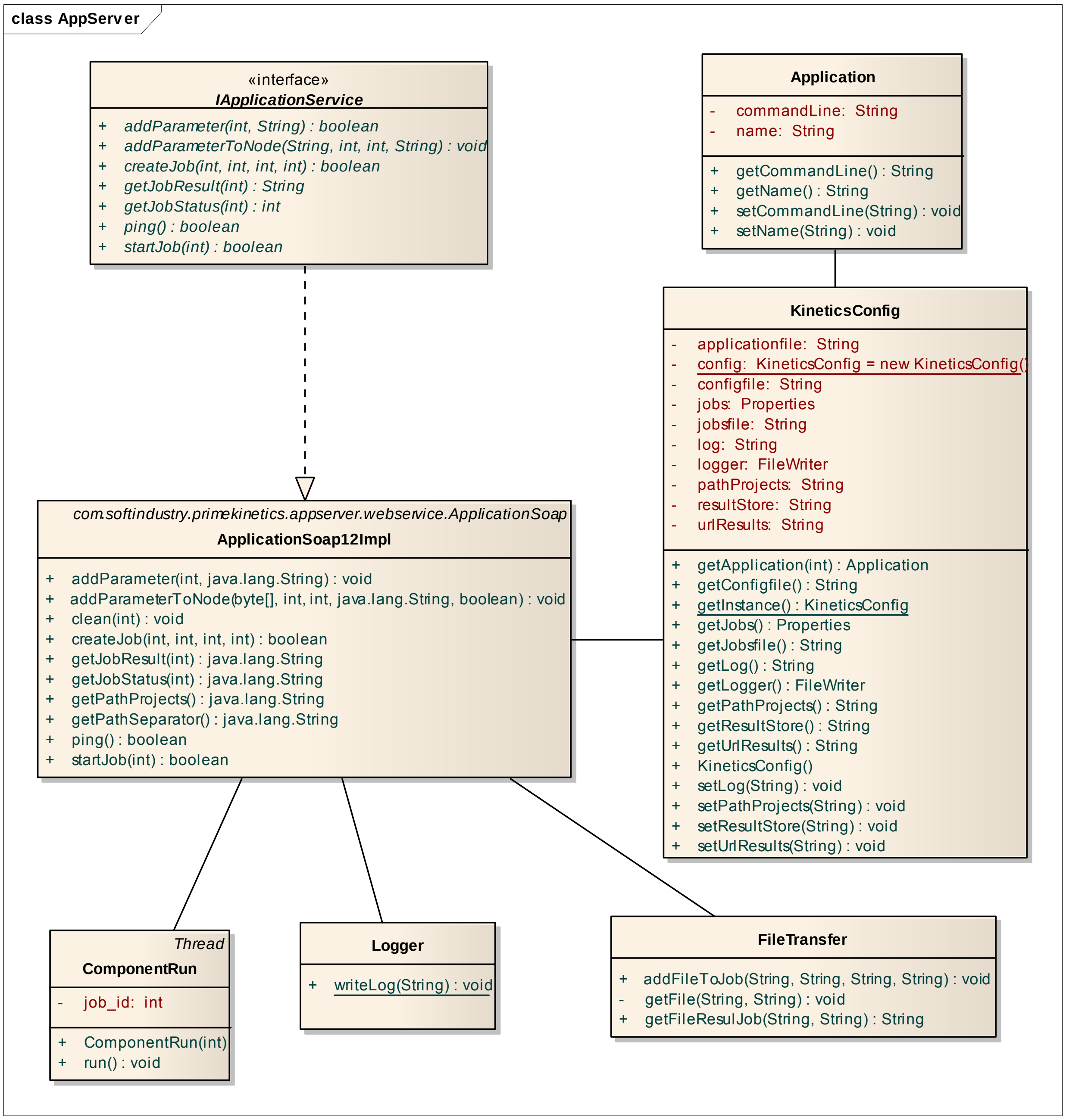

class AppServer
«interface»
IApplicationService
+ addParameter(int, String) : boolean
+ addParameterToNode(String, int, int, String) : void
+ createJob(int, int, int, int) : boolean
+ getJobResult(int) : String
+ getJobStatus(int) : int
+ ping() : boolean
+ startJob(int) : boolean
Application
- commandLine: String
- name: String
+ getCommandLine() : String
+ getName() : String
+ setCommandLine(String) : void
+ setName(String) : void
KineticsConfig
- applicationfile: String
- config: KineticsConfig = new KineticsConfig()
- configfile: String
- jobs: Properties
- jobsfile: String
- log: String
- logger: FileWriter
- pathProjects: String
- resultStore: String
- urlResults: String
+ getApplication(int) : Application
+ getConfigfile() : String
+ getInstance() : KineticsConfig
+ getJobs() : Properties
+ getJobsfile() : String
+ getLog() : String
+ getLogger() : FileWriter
+ getPathProjects() : String
+ getResultStore() : String
+ getUrlResults() : String
+ KineticsConfig()
+ setLog(String) : void
+ setPathProjects(String) : void
+ setResultStore(String) : void
+ setUrlResults(String) : void
com.softindustry.primekinetics.appserver.webservice.ApplicationSoap
ApplicationSoap12Impl
+ addParameter(int, java.lang.String) : void
+ addParameterToNode(byte[], int, int, java.lang.String, boolean) : void
+ clean(int) : void
+ createJob(int, int, int, int) : boolean
+ getJobResult(int) : java.lang.String
+ getJobStatus(int) : java.lang.String
+ getPathProjects() : java.lang.String
+ getPathSeparator() : java.lang.String
+ ping() : boolean
+ startJob(int) : boolean
Thread
ComponentRun
- job_id: int
+ ComponentRun(int)
+ run() : void
Logger
+ writeLog(String) : void
FileTransfer
+ addFileToJob(String, String, String, String) : void
- getFile(String, String) : void
+ getFileResulJob(String, String) : String

Figure 21. Application Server class diagram

IApplicationService—the interface describes all the web methods

Method	**Description**
addParameter	Receives the input information for a specified node
addParameterToNode	Receives the files, which contain the work results of nodes connected with the specified nodes
Clean	Clears the catalogue of the specified node
createJob	Creates the new job on the server
getJobResult	Receives the work result of the indicated job
getJobStatus	Receives the work status of the specified job
getPathProjects	Returns the path to the catalogue, in which all the files of current projects are stored
getPathSeparator	Returns the file separator for current OS("/" for Unix or "\" for Windows)
startJob	Executes the indicated job

Application—Contains the information about the scientific application

Method	**Description**
getCommandLine	Returns the command line, which will start the applications
getName	Returns the name of the scientific application
setCommandLine	Sets the command line
setName	Sets the application name

ComponentRun— Used for starting the scientific application. Each application is started in a separate flow

Method	**Description**
Componentrun	The constructor that receives as an input parameter job identifier
Run	Starts the scientific application

KineticsConfig—The class that provides the access to the main configuration files

Method	**Description**
getApplication	Returns the application by identifier
getConfigFile	Returns the path to the main configuration file
getJobsFile	Returns the path to the file, where the identifiers of current jobs are stored
getPathProjects	Returns the path to the catalogue, where the information of current projects is stored
getUrlResult	Returns the url, where the applications work results will be stored
getLogger	Returns the url on Logger class, which can be used for logs

8.9 Backend structure structure

Core library in the server is used in executing remote applications. In this library the authorization, authentication, and database work are managed. The main structure of the library is represented in Figure 22 and consists of the following parts:

Authentication module: Provides the site's users authorization and authentication service on the application server.

1. *Application Service*—The service used to process scientific application requests.
2. *JobService*—The service used to process current jobs and track execution status.
3. *ComponentService*—The service for component downloads.
4. *WorkflowService*—The service for work with the workflow.

8.9.1 Main modules

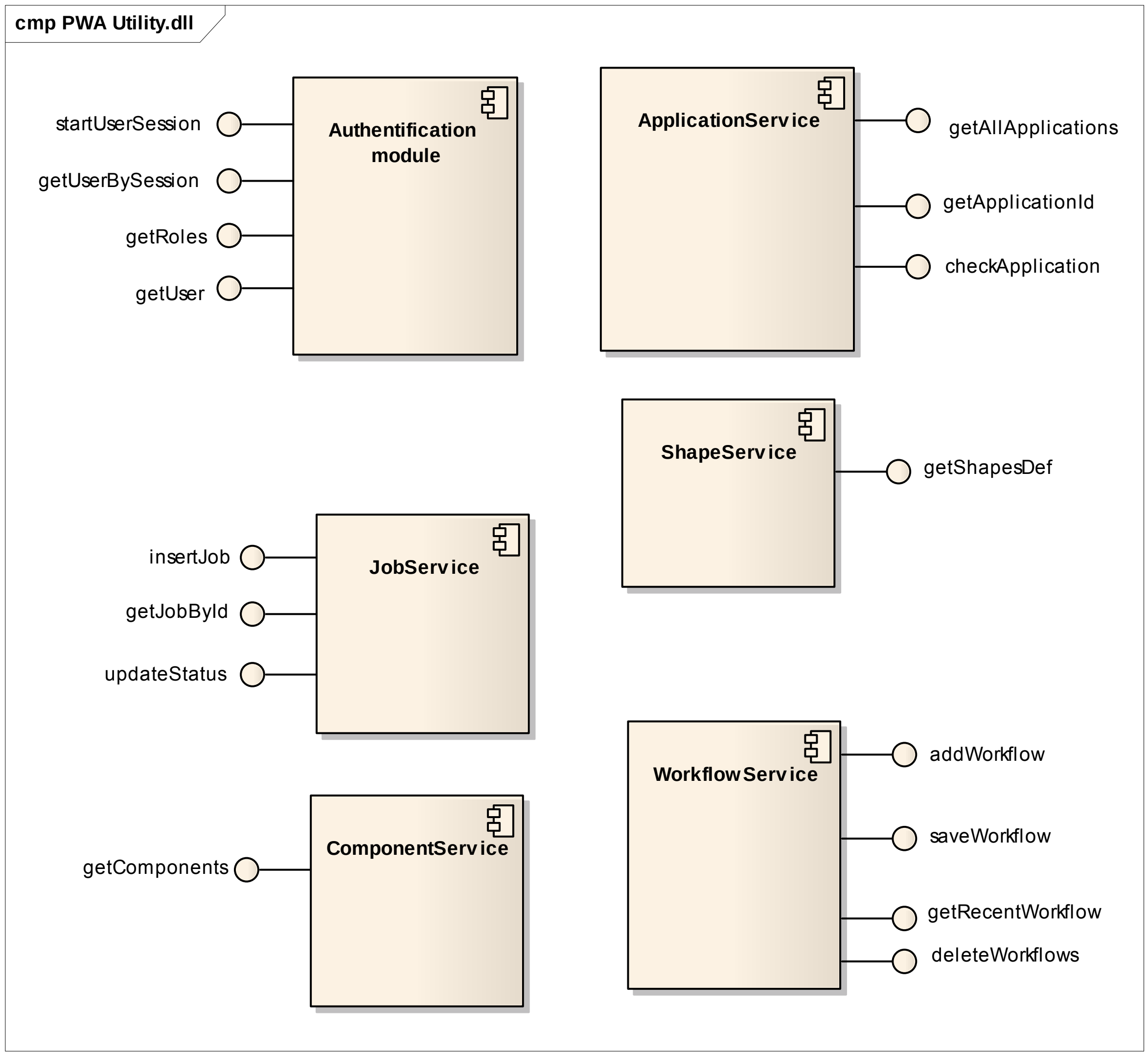


Figure 22. PWA backend main modules

8.9.2 Classes diagram

The main library classes are presented in Figure 23.

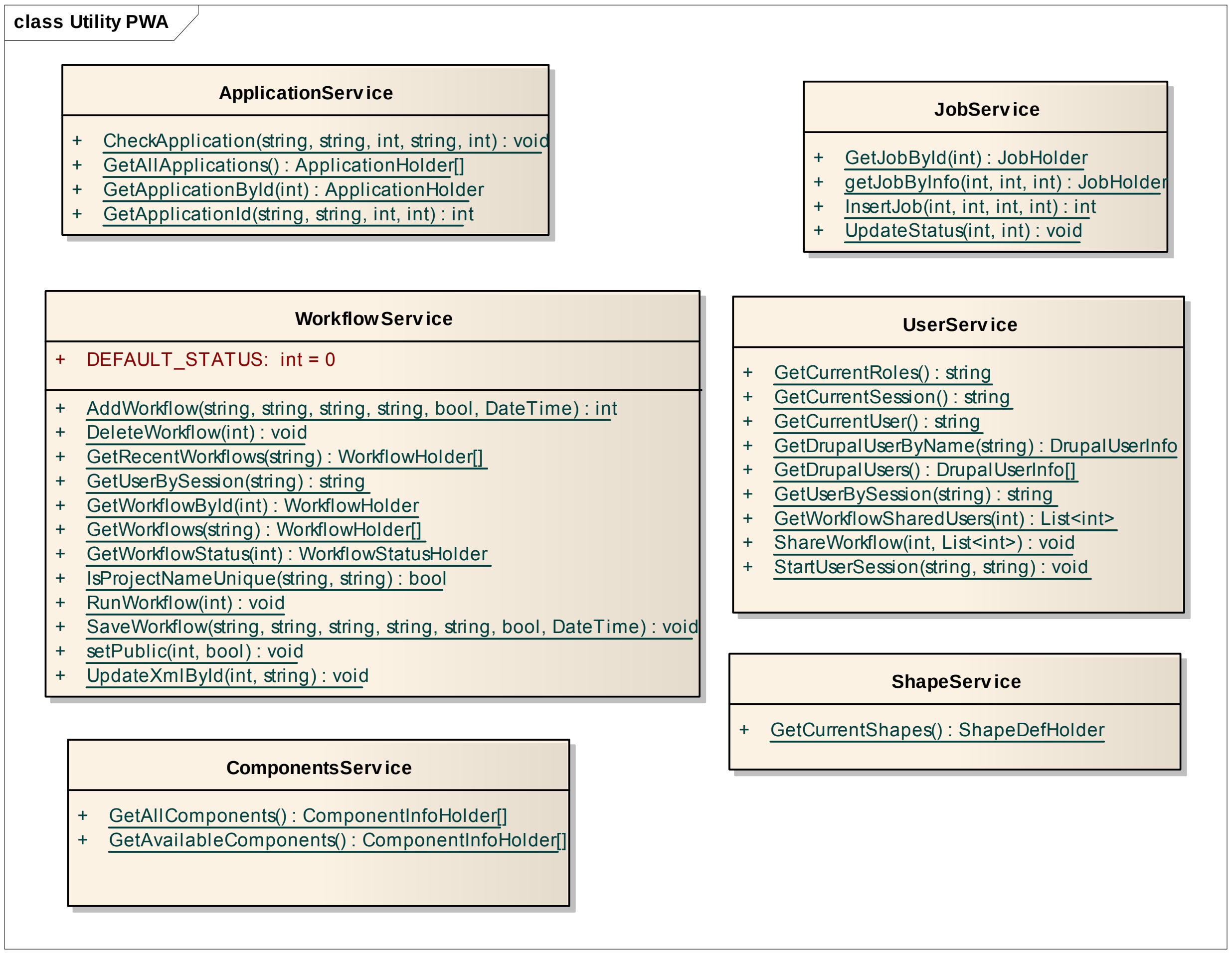

class Utility PWA
ApplicationService
+ CheckApplication(string, string, int, string, int) : void
+ GetAllApplications() : ApplicationHolder[]
+ GetApplicationById(int) : ApplicationHolder
+ GetApplicationId(string, string, int, int) : int
JobService
+ GetJobById(int) : JobHolder
+ getJobByInfo(int, int, int) : JobHolder
+ InsertJob(int, int, int, int) : int
+ UpdateStatus(int, int) : void
WorkflowService
+ DEFAULT_STATUS: int = 0
+ AddWorkflow(string, string, string, string, bool, DateTime) : int
+ DeleteWorkflow(int) : void
+ GetRecentWorkflows(string) : WorkflowHolder[]
+ GetUserBySession(string) : string
+ GetWorkflowById(int) : WorkflowHolder
+ GetWorkflows(string) : WorkflowHolder[]
+ GetWorkflowStatus(int) : WorkflowStatusHolder
+ IsProjectNameUnique(string, string) : bool
+ RunWorkflow(int) : void
+ SaveWorkflow(string, string, string, string, string, bool, DateTime) : void
+ setPublic(int, bool) : void
+ UpdateXmlById(int, string) : void
UserService
+ GetCurrentRoles() : string
+ GetCurrentSession() : string
+ GetCurrentUser() : string
+ GetDrupalUserByName(string) : DrupalUserInfo
+ GetDrupalUsers() : DrupalUserInfo[]
+ GetUserBySession(string) : string
+ GetWorkflowSharedUsers(int) : List<int>
+ ShareWorkflow(int, List<int>) : void
+ StartUserSession(string, string) : void
ShapeService
+ GetCurrentShapes() : ShapeDefHolder
ComponentsService
+ GetAllComponents() : ComponentInfoHolder[]
+ GetAvailableComponents() : ComponentInfoHolder[]

Figure 23. Utility.dll classes diagram

ApplicationService—The service for storing and receiving information about the scientific applications

Method	**Description**
CheckApplication	Checks whether the specified application exists in the database or not
GetAllApplication	Receives all the available applications
GetApplicationById	Returns the applications by the identifier
GetApplicationById	Returns the application by name, host, and the port on which it functions

JobService—The service for the work with the current tasks on applicationServer

Method	**Description**
GetJobById	Receives a job by the identifier
insertJob	Adds a new job
UpdateStatus	Updates the status of a job
getJobByInfo	Returns a job by project identifier, node and application

UserService—The service for work with users is implemented in this class

Method	**Assignment**
GetCurrentRoles	Returns the current user roles
GetCurrentSession	Returns the session of the current user
GetDrupalUsers	Returns all the users who are registered in the system
GetUserBySession	Returns the users on the session
StartUserSession	Starts the new session for a specified user

ShapeService—This class is used to download the shapes to PWA

Method	**Assignment**
GetCurrentShapes	Returns the shapes set as the xml-description

ComponentService—This class is used to download a client's component

Method	**Assignment**
GetAllComponents	Returns the available components

WorkflowService—This class provides the work with diagrams

Method	**Assignment**
AddWorkflow	Adds the new diagram
DeleteWorkflow	Deletes the diagram
GetWorkflowById	Retrieves the diagram by the identifier
SaveWorkflow	Saves the changes in the diagram
setPublic	Makes the diagram available for system users
getRecentWorkflows	Retrieves all the available diagrams

8.9.3 Database structure

The database structure is represented in Figure 24.

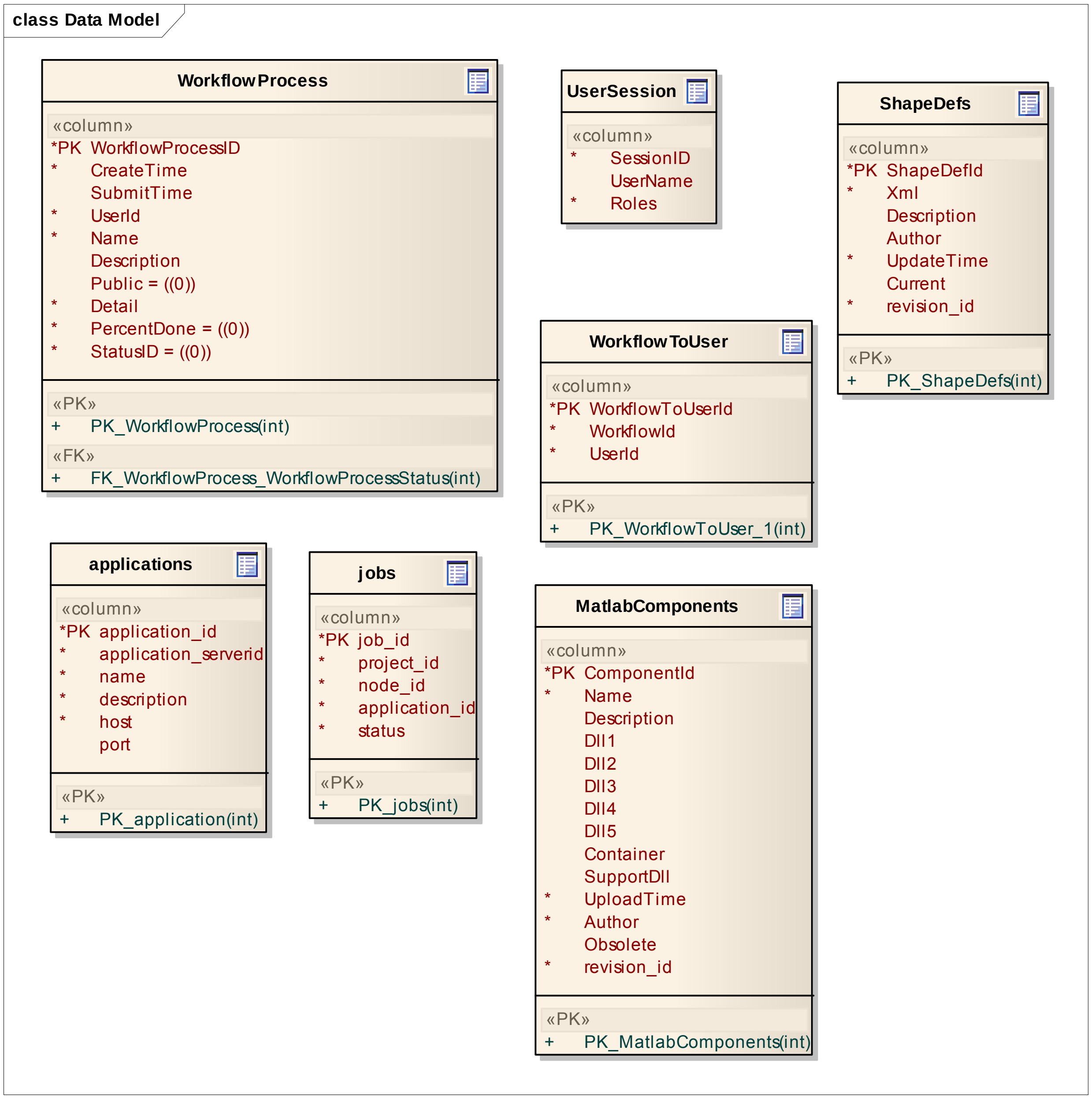


Figure 24. Database structure

ShapeDefs—Information about the shapes description

Field	**Description**
ShapeDefId	Shape description Identifier
Xml	Xml description of the shape
Description	Shape description
Author	The author who created the shape
updateTime	The update time
Revision_id	The revision identifier, to which the shapes refer

MatLabComponents—Information about the components

Field	**Description**
ComponentId	Component identifier
Name	Component name
Description	Component description
Dll1, Dll2, Dll3, Dll4, Dll5	Libraries names
SupportDll	Library name, which provides the interface of the connection with the PWA
Revision_id	Revision identifier, to which the component is referring
Author	The author who created the component
Obsolete	The indication that the component is out of date
UploadTime	The component download time

WorkflowProcess—Information about the created projects

Field	**Description**
WorkflowProcessId	Unique project identifier
CreateTime	Creation date
SubmitTime	The last starting time
UserId	The user identifier who created the project
Name	Project name
Description	Project description
Public	The indicator that the project is available for all the users

UserSession—In this table the unique line that identifies the user's session is stored

Field	**Description**
SessionId	Session identifier
UserName	User's login
Roles	User's roles

Applications—Information about the servers' applications registered in the system

Field	**Description**
applicationId	Unique identifier of the application server
Application_serverid	The identifier of the applications on the server
Name	Name
Description	Description
Host	IP address or DNS host name
Port	The port on which the web-server is working

Jobs—Information about the current tasks on application servers

Field	**Description**
Job_id	Job unique identifier
Project_id	Project identifier
Node_id	The node identifier on the diagram
Application_id	Applications identifier
Status	Implementation status

WorkflowToUser—Information about the user's access to the project

Field	**Description**
WorkflowToUserId	Unique identifier
WorkflowId	Unique project identifier
UserId	User identifier

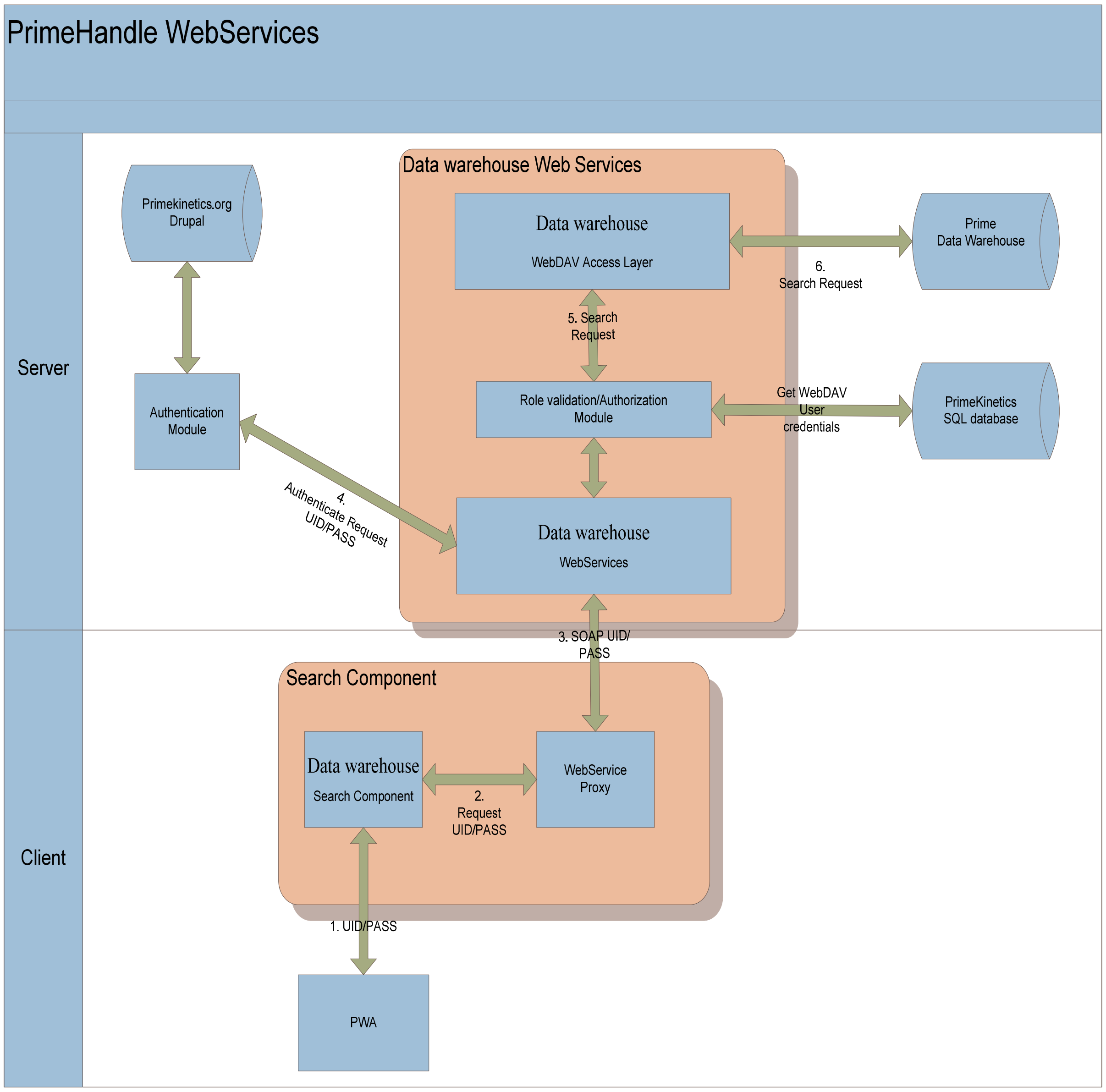
8.10 Web service Description

The PrIMeKineticsClient.dll library interacts with the server by means of the web service. The main methods used are represented below:

Method	**Description**
GetWorkflow	Returns the workflow by identifier
DeleteWorkflow	Delete the workflow
GetAvailableComponents	Retrieves all the available components
GetCurrentShapes	Retrieves the current shapes
GetWorkflowSharedUsers	Retrieves the users, who have the access to the project
ShareWorkflow	Sets the rights for the project access of specified users
SetPublicWorkflow	Makes the project public
insertJob	Saves a new job on the server
updateJobStatus	Updates the job status
clearProduction	Deletes all components and resources
uploadProduction	Uploads the new resources, components, and shapes

9 PrimeHandle Web Services

PrimeHandle web services are implemented to allow easy access from PWA components to prime data warehouse repository. PrimeHandle web services incapsulate WebDAV interface and enables PWA components to query and update prime data warehouse via web services. See the architectual diagram for more details:


PrimeHandle WebServices
Server
Client
Primekinetics.org Drupal
Authentication Module
Data warehouse Web Services
Data warehouse WebDAV Access Layer
5. Search Request
Role validation/Authorization Module
Data warehouse WebServices
4. Authenticate Request UID/PASS
6. Search Request
Prime Data Warehouse
Get WebDAV User credentials
PrimeKinetics SQL database
3. SOAP UID/PASS
Search Component
Data warehouse Search Component
2. Request UID/PASS
WebService Proxy
1. UID/PASS
PWA

9.1 PrimeHandle Web Service methods

Below is the list of currently implemented prime data warehouse methods:

Copy

Copy method can be used to copy a file from the WebDAV source path to the destination path.

Usage: Copy(string sourcePath, string destPath, string login, string password)

Example:
Copy('depository/experiments/catalog/x00000001.xml','depository/experiments/catalog/_attic/x00000001_0.xml','Username','Password')

Delete

Delete method can be used to delete a WebDAV file.

Usage: Delete(string strPath, string login, string password)

Example: Delete('depository/experiments/catalog/x00000001.xml','Username','Password')\

Exist

Exist method is to check if an XML file or a directory exists on a specified WebDAV path.

Usage: Exist(string strPath, string login, string password)

Example: Exist('depository/experiments/catalog/x00000001.xml','Username','Password')

GetDetails

GetDetails method is to get the details of a WebDAV XML file.

Usage: GetDetails(string strPath, string login, string password)

Example:
GetDetails('depository/experiments/catalog/x00000001.xml','Username','Password')

GetFile1

GetFile1 method can be used to download a WebDav file.

Usage: GetFile1(string strPath, string login, string password)

Example: GetFile1('depository/experiments/catalog/x00000001.xml','Username','Password')

GetPropertyNames

GetPropertyNames method is to list property names of the specified WebDAV file.

Usage: GetPropertyNames(string strPath, sstring login, string password)

Example:
GetPropertyNames('depository/experiments/catalog/x00000001.xml','Username','Password')

GetXml

GetXml method is to get the XML description of the specified WebDAV XML file.

Usage: GetXml(string strPath, string login, string password)

Example: GetXml('depository/experiments/catalog/x00000001.xml','Username','Password')

Move

Move method can be used to move a file from the WebDAV source path to the destination path.

Usage: Move(string sourcePath, string destPath, string login, string password)

Example: Move('depository/experiments/catalog/x00000001.xml','depository/experiments/catalog/_attic/x00000001_0.xml','Username','Password')

PropFind

PropFind method is to get a property of the specified WebDAV file.

Usage: PropFind(string strPath, string propName, string login, string password)

Example: PropFind('depository/experiments/catalog/x00000001.xml','getlastmodified','Username','Password')

PropPatch

PropPatch method can be used to set a property of the specified WebDAV file.

Usage: PropPatch(string strPath, string propName, string propValue, string login, string password)

Example: PropPatch('depository/experiments/catalog/x00000001.xml','submittedBy','submitter','Username','Password')

Search

Search method can be used to search WebDAV database.

Usage: Search(string collectionPath, string searchArg, string depth, string login, string password)

Example: Search('depository/experiments/catalog','CONTAINS('shock')','DEEP','Username','Password')

Submitfile

Submitfile method is to submit file byte[] to a specified WebDAV path, and propatch username and reason.

Usage: Submitfile(byte[] buffer, string strPath, string newOrOld, string reason, string login, string password)

Example: Submitfile(buffer,'depository/bibliography/catalog/b00000000.xml', 'new', 'new file from Username', 'Username','Password')

Upload1

Upload1 method is to upload a file byte[] to a specified WebDAV path.

Usage: Upload1(byte[] buffer, string strPath, string login, string password)

Example:
Upload1(buffer,'depository/experiments/catalog/x00000001.xml','Username','Password')

ValidateXml

ValidateXml method can be used to check if an XML string is valid against Schema.

Usage: ValidateXml(string strXml, string login, string password)

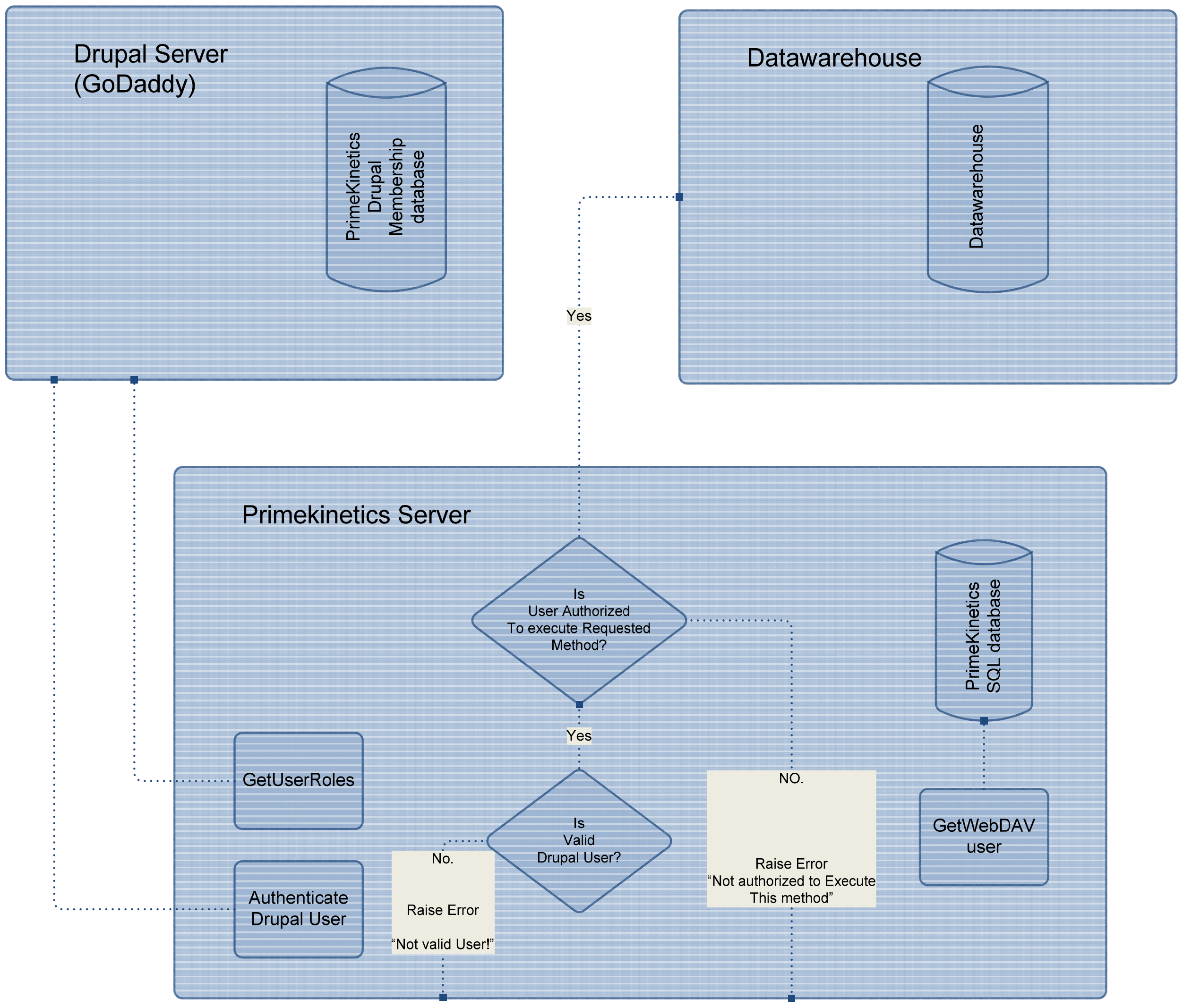
Example: TextReader tr = new StreamReader('C:/x00000001.xml');
String strXml = tr.ReadToEnd();
ValidateXml(strXml,'Username','Password')

The number of web service methods will expand in the future, so please use the following link to get the latest information on availalble prime data warehouse web service methods:

http://dispatcher.primekinetics.org/workflow_dev_test/services/PrimeHandle.asmx?wsdl

9.2 PrimeHandle Web Services authentication mechanism

PrimeKinetics.PrimHandle.dll was implemented to encapsulate PrimeHandle WebDAV authentication mechanism (see the digram above). PrimeKientics.PrimeHandle.dll is located on prime server and provides abstraction layer for PrimeHandle web service methods to authenticate on WebDAV datawarehose before they can gain access WebDAV data.



Drupal Server (GoDaddy)
PrimeKinetics Drupal Membership database
Datawarehouse
Datawarehouse
Yes
Primekinetics Server
Is User Authorized To execute Requested Method?
PrimeKinetics SQL database
Yes
GetUserRoles
NO.
Raise Error "Not authorized to Execute This method"
GetWebDAV user
Is Valid Drupal User?
No.
Raise Error "Not valid User!"
Authenticate Drupal User

WebDAV admin user login and password information is stored on the server in Prime SQL database and is configurable from primekinetics admin user interface.

See **Primehandle web interface configuration** section for configuration details

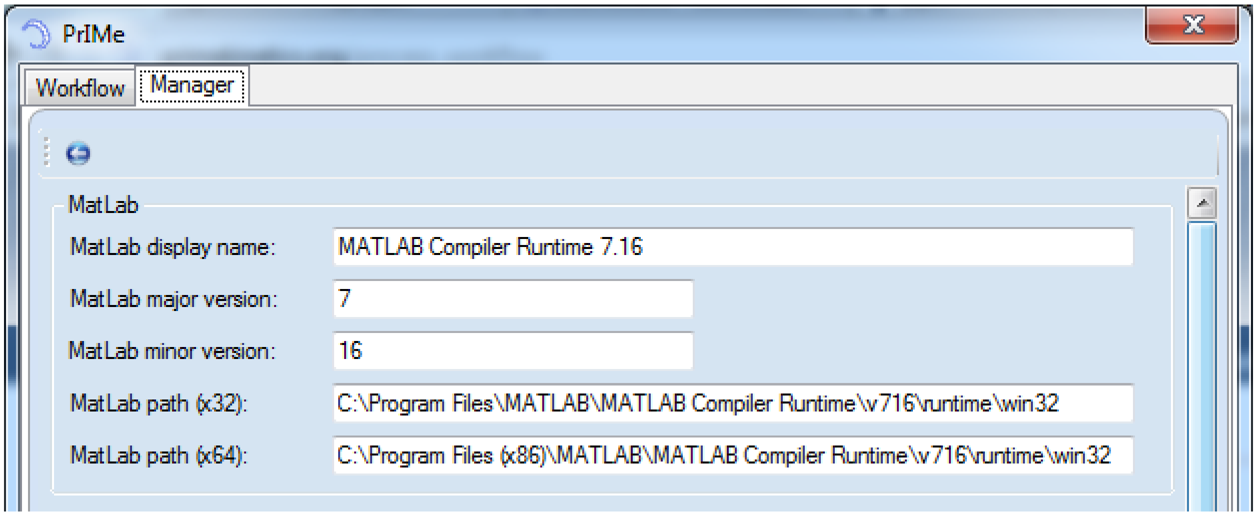
10 System Configuration and Maintenance

Matlab runtime version configuration

This user interface was designed to ease the process of upgrading the system to a latest Matlab version. To upgrade PWA to a new Matlab version the following steps have to be taken:

1. Obtain latest Matlab runtime executable and upload (FTP) it to PWA server to the following location: C:\Inetpub\wwwroot\workflow
2. Open System Preferences tab on PWA Component Uploader application http://dispatcher.primekinetics.org/workflow_dev/manager.aspx and configure following Matlab runtime parameters

- Matlab display name
- Matlab major version
- Matlab minor version
- Matlab path (x32) – location of Matlab runtime on 32-bit system
- Matlab path (x64) – location of Matlab runtime on 64-bit system



MatLab

MatLab display name:	MATLAB Compiler Runtime 7.16
MatLab major version:	7
MatLab minor version:	16
MatLab path (x32):	C:\Program Files\MATLAB\MATLAB Compiler Runtime\v716\runtime\win32
MatLab path (x64):	C:\Program Files (x86)\MATLAB\MATLAB Compiler Runtime\v716\runtime\win32

Primehandle web interface configuration

This user interface was developed to configure PWA for WebDAV system access. The following parameters have to be specified:

Prime server – WebDAV server URL

Prime host – WebDAV server name

Prime port – port number (optional) to access WebDAV server

Prime protocol – network protocol used to access WebDAV server

WebDAV user – WebDAV system user name

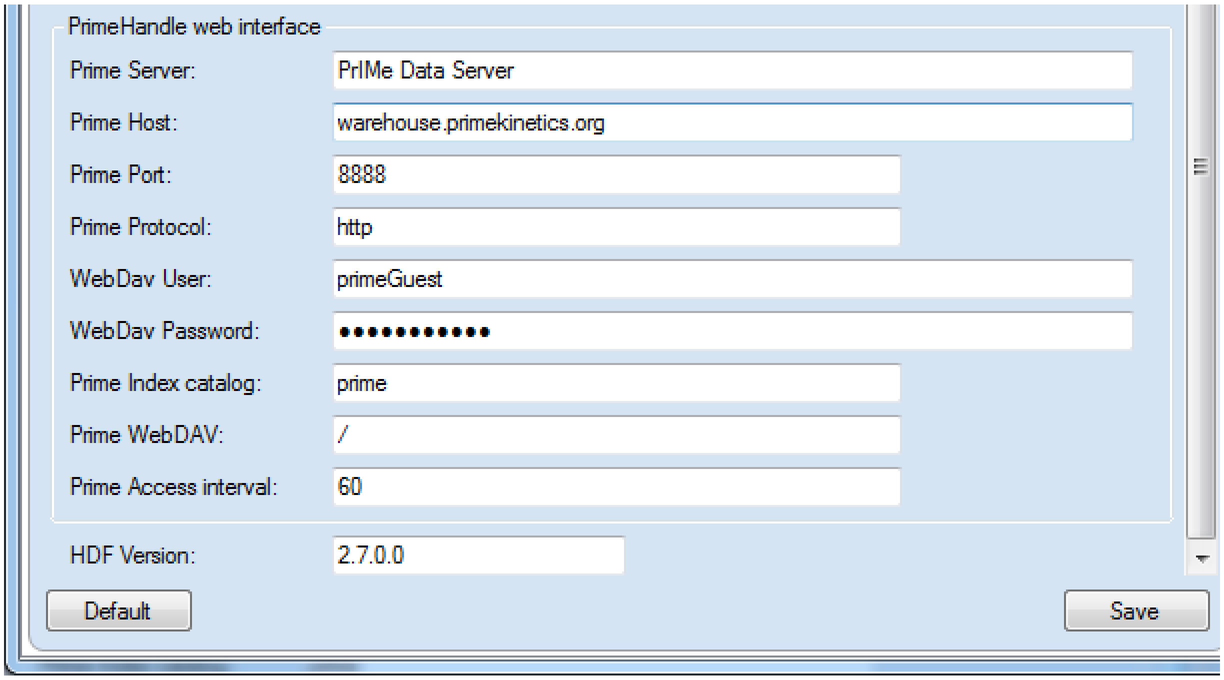
WebDav password – WebDAV system user password

Prime Index Catalog – WebDAV root catalog

Prime WebDav – not used

Prime Access Interval – not used

HDF Version – version of the HDF viewer



PrimeHandle web interface

Prime Server:	PrIMe Data Server
Prime Host:	warehouse.primekinetics.org
Prime Port:	8888
Prime Protocol:	http
WebDav User:	primeGuest
WebDav Password:	●●●●●●●●●●●●
Prime Index catalog:	prime
Prime WebDAV:	/
Prime Access interval:	60
HDF Version:	2.7.0.0

11 Technologies used

11.1 PrIMe Portal

The PrIMe portal is executed using the PHP language with the help of CMF Drupal-6. The standard modules of the Drupal core set are developed by third parties and obtained from the repository drupal.org. Part of the modules was modified specifically for the PrIMe portal.

The PrIMe portal uses MySQL for the database technology. It is working on the web server technology Apache2 under the OC Windows-2003 Server management.

11.2 Scientific Component Uploader and PrIMe Workflow Application

Both the SCU and the PWA utilize Microsoft .NET technologies. All codes are written in C#. To enable the feature of native code implementation in the context of the client's browser, Active X technology was used.

In the capacity of DBMS the MS SQL Server 2005 is used. It is run on the web server IIS under the management of the OC Windows-2003 Server.

11.3 Application server

The application server utilizes Java technologies. For the creation of web services the AXIS framework is used. It is run on the Tomcat 6 web server.

12 Personnel Supported

This project supported mainly the programming consultant, Michael Gutkin, a graduate student, Devin Yeates, along with the Principal Investigator, Professor Michael Frenklach.

13 Publications and Presentations

1. "Methodology and Infrastructure for Predictive Modeling", Korea University, Research Institute of Korean Studies, Seoul, Korea, October 18, 2010.
2. "Methodology and Infrastructure for Predictive Modeling", Swiss Federal Institute of Technology (ETH Zurich), Institute of Process Engineering, May 2, 2011.
3. "Is Your Experiment Informative?" Stanford University, Mechanical Engineering Department, High-Temperature Gas-Dynamics Laboratory, November 16, 2011.
4. "Uncertainty-Quantified Analysis of Complex Experimental Data," D. R. Yeates, W. Li, P. R. Westmoreland, T. Russi, A. Packard, and M. Frenklach, *Proceedings of the 7th U.S. National Combustion Meeting*, Atlanta, GA, 2011, Paper No. 2D01.
5. "UQ-Prediction on the Feasible Set," M. Frenklach, A. Packard, T. Russi, X. You, D. Yeates, W. Speight, M. Gutkin, 13th International Conference on Numerical Combustion, Corfu, Greece, April 27-29, 2011.
6. "Methodology and Infrastructure for Predictive Modeling," M. Frenklach, A. Packard, T. Russi, X. You, D. Yeates, W. Speight, and M. Gutkin, The 7th International Conference on Chemical Kinetics, MIT, Cambridge, MA, July 10-14, 2011.
7. "Process informatics tools for predictive modeling: Hydrogen combustion," X. You, A. Packard, and M. Frenklach, *Int. J. Chem. Kinet.* **44**, 101-116 (2012).

14 Significant Interactions

In collaboration with Professor Phillip Westmoreland's group, we applied the developed here tools to the analysis of data collected in a fuel-lean $C_2H_2/O_2/Ar$ premixed laminar flat flame, mapped with VUV-photoionization molecular-beam mass spectrometry at the Advanced Light Source of Lawrence Berkeley National Laboratory.